Collection Ressources Informatiques

JavaScript

Des fondamentaux aux concepts avancés

(bibliothèques Prototype et Script.aculo.us)

Emmanuel GUTIERREZ

Fichiers à télécharger
www.editions-eni.fr

INFORMATIQUE TECHNIQUE

eni éditions

JavaScript

Des fondamentaux aux concepts avancés

Emmanuel GUTIERREZ



Résumé

Ce livre sur **Javascript** est destiné à tous ceux qui se préoccupent de donner plus **d'interaction** à leurs sites web. Il vise deux objectifs : tout d'abord **maîtriser les fondements** de JavaScript afin d'élaborer les scripts les plus fréquemment utilisés sur le net, puis découvrir le **nouveau potentiel de JavaScript** aussi bien grâce à son utilisation avec les feuilles de styles en cascade (**CSS**), le **DHTML**, **AJAX** qu'avec les bibliothèques telles que **Prototype** ou **Script.aculo.us**.
Après avoir présenté la **syntaxe de base**, le livre prend appui sur des exemples significatifs (gestion des formulaires, du temps, des menus de navigation, glissé-déposé, autocomplétion), pour les commenter et démontrer l'omniprésence de JavaScript dans une **architecture Web 2.0**.
Le livre accompagne le lecteur tout au long d'un véritable parcours allant des rudiments de JavaScript jusqu'à la découverte des concepts les plus avancés.
Les exemples de script cités dans l'ouvrage sont en téléchargement sur cette page.

L'auteur

Après plusieurs années passées en tant que formateur, **Emmanuel Gutierrez** est aujourd'hui consultant informatique, gérant d'un centre de formation qu'il a créé. Ses différentes missions en entreprises autour de la création de sites web s'allient à son expérience pédagogique pour fournir au lecteur un ouvrage réellement opérationnel pour maîtriser le développement en JavaScript.

Historique et versions de JavaScript

Le moins que l'on puisse dire c'est que JavaScript est un langage très controversé. Au début de l'Internet, les pages étaient constituées uniquement de textes et de liens hypertextes, ce qui restreignait l'usage à des scientifiques et des universitaires. De toutes manières, les contraintes techniques de l'époque, notamment au niveau des débits de transfert, ne pouvaient pas proposer autre chose de mieux. C'est au milieu des années 1990 que le besoin de disposer de sites Internet plus conviviaux et proposant plus de services est apparu. Brendan Eich, alors ingénieur informaticien chez Netscape est chargé du développement d'un nouveau navigateur web. Il en profite pour développer un langage de Script, à l'origine nommé LiveScript, et qui devait être un complément à Java (ces deux langages sont souvent confondus du fait de leur appellation quasi-identique, bien qu'ils n'aient que peu de choses en commun). L'objet de ce langage de script est de rendre les pages Internet plus attractives, plus conviviales pour le visiteur, en permettant davantage de choses sans pour autant faire appel à la programmation côté serveur. Pour ce faire, le navigateur doit pouvoir interpréter le code JavaScript. Netscape décide d'implémenter nativement LiveScript dans la version 2.0 de son navigateur (alors baptisé Netscape Navigator) dès 1995. Débute alors, une période de grande popularité des langages de script et Microsoft ne pouvait que se résoudre à sortir sa propre version. Ce fut Jscript sorti en 1996 et intégré à son navigateur Internet Explorer dont la dernière version est aujourd'hui Jscript.Net. Les versions de JavaScript se sont alors succédées, apportant pour chacune d'entre elles son lot d'améliorations. Tout le monde a pu constater que l'Internet a rapidement été envahi de pages comportant de petits scripts permettant, par exemple, d'afficher l'heure, la date, le nom du visiteur, ou effectuant la validation du contenu de champs de formulaire. Cependant, même si JavaScript suit les instructions données par l'ECMA (*European Computer Manufacturers Association*), organisme international chargé de la standardisation des systèmes d'information et de communication, les éditeurs de logiciels (Microsoft d'un côté avec Internet Explorer et Sun de l'autre avec Firefox), ont élaboré, depuis le début, des navigateurs qui implémentent différemment JavaScript. De ce fait, certains scripts peuvent très bien s'exécuter normalement sur un navigateur et paradoxalement, générer une erreur sur un autre. C'est en partie à cause de cela qu'à la fin des années 1990, d'autres langages tels ASP ou PHP deviendront plus populaires. Mais c'est surtout l'utilisation à outrance de pop-up (fenêtre surgissante) qui est à l'origine de la baisse d'intérêt pour l'emploi de JavaScript. Leur prolifération a énormément nui à JavaScript et l'exaspération des utilisateurs a fini par en cacher les avantages aux yeux des développeurs ; certains allant même jusqu'à le considérer comme un sous-langage. Heureusement, l'arrivée des bloqueurs de pop-up intégrés aux navigateurs a permis à JavaScript de redorer son blason.

Le tableau suivant permet de lier la version de JavaScript avec celle du navigateur :

Année de sortie	Version de JavaScript	Navigateurs compatibles
1995	1.0	Internet Explorer 3.0 Netscape Navigator 2.0
1996	1.1	Internet Explorer 4.0 Netscape Navigator 3.0
1997	1.2	Internet Explorer 4.0 Netscape Navigator 4.0
1998	1.3	Netscape Navigator 4.5
1999	1.4	Internet Explorer 6.0 Netscape Navigator 6.0
2000	1.5	Netscape Navigator 6.0
2005	1.6	Firefox 1.0
2006	1.7	Firefox 2.0 Internet Explorer 7.0

Aujourd'hui, JavaScript est redevenu un langage à la mode. Il est d'ailleurs intéressant de constater que les mêmes personnes, qui, il y a quelques temps, décriaient ce langage, l'utilisent aujourd'hui à tort et à travers. En effet, l'arrivée de nouvelles technologies web et notamment du web 2.0, redonne au développement JavaScript une place centrale, notamment par son utilisation conjointe avec XML ou par son utilisation asynchrone (Ajax), dont nous reparlerons plus tard. Ceci dit, il n'est pas inutile de pointer du doigt les avantages et les limites de l'utilisation de JavaScript.

Limites et avantages de JavaScript

Comme nous l'avons déjà précisé, JavaScript est un langage universel qui se retrouve dans de nombreuses pages HTML, en complément de ce code. Grâce à JavaScript, les pages HTML sont plus riches et disposent de nombreuses fonctionnalités supplémentaires.

Savoir rédiger des scripts en JavaScript, c'est permettre aux visiteurs de vos pages HTML d'accéder à d'autres fonctionnalités, à d'autres services et d'améliorer ainsi la professionnalisation d'un site. Ainsi, il y a encore quelques temps, lorsqu'un utilisateur choisissait un identifiant pour la première fois, il fallait cliquer sur un bouton et attendre une réponse de la part du serveur. Et, celui-ci, parfois, invitait à recommencer le processus de création, le pseudo étant déjà pris. Alors qu'aujourd'hui, avec l'emploi de la technologie AJAX, le contrôle s'effectue en arrière-plan pendant que le visiteur complète sa fiche. Il est indéniable que JavaScript contribue fortement à la convivialité d'un site Internet et améliore, par conséquent, la fidélisation des visiteurs.

Étant donné cette large diffusion, savoir rédiger des scripts JavaScript est devenu, aujourd'hui, une connaissance de base de tout développeur web.

Pour autant, l'usage de JavaScript n'est pas réservé au web, en effet plusieurs logiciels du marché tels Adobe Photoshop ou Adobe Illustrator utilisent des variantes très proches de JavaScript pour automatiser certaines tâches. En ce qui concerne la difficulté du langage, certains pourraient être réticents à l'analyse des pages HTML contenant du code JavaScript, mais finalement avec un peu de temps et de recherche, JavaScript est un langage dont la maîtrise est assez facile. D'autant plus si vous êtes déjà familier avec d'autres langages tels le Visual Basic ou le langage C, par exemple, même si une adaptation à quelques particularités est tout de même nécessaire.

À l'inverse, JavaScript ne peut pas tout faire. Comme c'est un langage de script qui s'exécute côté client, il lui est impossible de s'interfacer avec une base de données de type Mysql ou SQL, par exemple. Pour remplir cette mission, il faut impérativement passer par de la programmation côté serveur avec des langages tels ASP ou PHP. Autre point important, JavaScript n'est pas en mesure d'écrire ou de lire sur le disque dur du poste client (hormis les cookies qui ne sont que de petits fichiers texte et dont nous traiterons les différents aspects au chapitre Améliorer l'interactivité avec JavaScript et CSS). Cette dernière limitation n'en constitue pas pour autant un défaut, car ainsi JavaScript ne propage pas d'infections virales fortement dangereuses.

Une autre particularité de JavaScript réside dans le fait qu'il ne nécessite pas d'éditeur particulier ni de compilateur. Il est très facile de rédiger des scripts directement dans le code de la page HTML en passant par un simple éditeur de texte de type Wordpad ou un éditeur de code HTML. Il existe, cependant, des outils de conception dont l'apport n'est pas négligeable.

Outils de conception

Les outils permettant d'insérer du code JavaScript sont nombreux. Cela va du simple logiciel éditeur de texte, comme par exemple WordPad de Windows, à l'outil spécifique Aptana Studio, en passant par les éditeurs de code HTML tels Dreamweaver ou FrontPage, avec lesquels il est possible d'insérer des blocs JavaScript. L'usage de ces logiciels permet de disposer d'un certains nombre d'outils facilitant l'écriture du code. Il est, par exemple, fort simple de :

- vérifier une syntaxe par la coloration automatique du code ;
- disposer de l'autocomplétion (proposition des méthodes ou propriétés disponibles de l'objet) ;
- connaître la valeur d'une variable lors de l'exécution d'un script.

Pour cela, vous pouvez opter pour un logiciel tel Aptana que vous pouvez télécharger à l'adresse http://www.aptana.com. Malheureusement cet IDE (environnement de développement) est en anglais, ce qui peut être décourageant. Il s'avèrera cependant utile pour le débogage au même titre que d'autres outils dont nous reparlerons aux chapitres suivants.

Votre outil de conception sélectionné, il convient de construire un environnement de développement et de tests afin de perdre le minimum de temps lors de la recherche des erreurs, qui surviendront inévitablement.

Paramétrages et environnement optimal de test

Il faut avoir à l'esprit que pour débuter en JavaScript, un minimum de connaissances en HTML est nécessaire et notamment la notion de balise permettant de se situer dans la page. Pour mémoire, nous rappellerons simplement qu'une page HTML se divise en deux grandes parties :

- la partie head (tête en français) dans laquelle se situent les informations correspondant à la description du contenu ;
- la partie body (corps en français) où figure le code permettant la construction des objets dans la page (champs de formulaire, zone de texte, image, etc.).

Un script JavaScript peut se trouver au choix dans l'une ou l'autre de ces deux parties. Cependant, par convention, les scripts se retrouvent plus fréquemment dans la partie head de la page. Leur exécution peut alors être immédiate (au chargement de la page) ou différée (clic sur un bouton, par exemple). Il faudra, dans ce cas, utiliser la programmation évènementielle et les fonctions pour que le code s'exécute. Ces points sont traités au chapitre Fonctions et évènements du présent ouvrage. La position des scripts dans la partie head ne signifie pas pour autant que les scripts seront indexés par les moteurs de recherche. En effet, jusqu'à présent, les moteurs tels Google ou Yahoo ne proposent pas de références à partir de ces éléments de code, mais avec le développement des technologies du web 2.0, ils le feront un jour ou l'autre. Pour l'instant, dans le cas où une page contient des liens à l'intérieur d'un menu écrit à l'aide de JavaScript, ces adresses ne seront pas référencées par les robots. Il est donc fortement recommandé d'ajouter les liens en HTML à l'aide de la balise `<a href="mapagevisibleparlesmoteurs.html">`. Après avoir détaillé la position des scripts JavaScript, il faut éclaircir la manière de les insérer. Nous avons déjà vu que JavaScript ne nécessitait pas d'environnement particulier. Il suffit simplement d'une page HTML à l'intérieur de laquelle vous ajoutez des lignes rédigées en JavaScript entre deux balises. La première balise indique au navigateur le début du script JavaScript et la seconde en indique la fin. Les deux balises à utiliser sont les suivantes :

Au début du script :

```
<script language="javascript">
```

et à la fin du script

```
</script>
```

Entre les deux balises, le nombre de lignes de code est illimité. Il est possible d'ajouter la version de JavaScript utilisée et d'obtenir une ligne de type :

```
<script language="javascript 1.5">
```

Cependant, les navigateurs s'accommodent très bien de la première syntaxe et, en fonction de leur version, s'adaptent à la version JavaScript. De plus, en spécifiant une des dernières versions, vous contraignez les visiteurs à disposer des derniers navigateurs, ce qui limite la portée de votre code. Si, malgré tout, vous souhaitez faire passer des informations aux visiteurs ayant un navigateur ne supportant pas JavaScript, il faut le faire entre les balises <noscript> et </noscript> que l'on place juste après.

```
<script language="javascript">
document.write ("bonjour");
</script>
<noscript> votre navigateur ne peut pas lire le code JavaScript</noscript>
```

Dans cet exemple, le navigateur affichera Bonjour s'il est compatible ou le texte figurant entre les balises <noscript> et </noscript> s'il ne l'est pas.

Une fois les deux balises insérées et le code écrit, il suffit d'enregistrer vos pages avec un nom différent par le menu **Fichier - Enregistrer sous**. Ainsi, vous pouvez constituer progressivement une bibliothèque de pages HTML contenant des scripts JavaScript à réemployer lors d'utilisations bien spécifiques.

En d'autres termes, il vous faut, pour débuter, une page HTML rudimentaire qui vous servira de matrice pour toutes les autres pages contenant les scripts.

HTML et JavaScript

Nous avons vu précédemment que JavaScript et HTML étaient intimement liés l'un à l'autre, le code HTML servant généralement de conteneur au bloc d'instructions JavaScript. Après chargement du code HTML, le navigateur exécute les blocs d'instructions JavaScript et permet ainsi d'enrichir la page de nouvelles fonctionnalités. Pourtant, il existe un autre type d'exécution des scripts JavaScript.

Les deux types d'exécution du code JavaScript

En effet, les blocs de script JavaScript peuvent être directement présents dans le code HTML de la page entre deux balises (une de début et une de fin), ou écrits dans un fichier JavaScript au format .js, totalement dissocié du code HTML de la page. Le premier cas est désigné sous le terme de JavaScript interne par opposition au second, appelé JavaScript externe.

Aucun des deux types d'exécution n'est meilleur que l'autre, il s'agit simplement d'une préférence de mode de développement. La deuxième solution permet toutefois de réutiliser le code dans d'autres pages HTML sans qu'il soit nécessaire de le réécrire ou de faire un copier coller.

Concrètement, le script JavaScript est rédigé dans un fichier particulier à l'aide d'un éditeur de texte, par exemple, et doit être enregistré sans formatage au format .js. Si l'éditeur de texte ne propose pas par défaut cette extension, il suffit de l'ajouter lors de la sauvegarde du fichier. Afin de pouvoir retrouver facilement le rôle de votre script, il convient de le nommer avec un nom explicite.

Par la suite, il est très simple d'appeler le fichier JavaScript dans la page HTML en respectant la syntaxe suivante :

```
<script src="fichier_javascript.js"></script>
```

Bien sûr, le fichier devra être présent dans le même répertoire de votre disque dur ou du serveur que le fichier HTML appelant.

Par souci de simplification, tous les exemples de cet ouvrage sont conçus en JavaScript interne, ainsi vous retrouverez plus facilement l'ensemble du code.

En plus de ces règles purement liées à l'organisation des scripts, il existe un certain nombre de règles de syntaxe à respecter.

Les règles de syntaxe du code

Pour bien débuter en JavaScript le respect de ces règles est fondamental, car ce langage est peu souple et n'autorise pas d'erreurs comme vous le constaterez dans les paragraphes suivants.

1. La casse

Une des principales difficultés de JavaScript réside dans le fait que c'est un langage sensible à la casse, c'est-à-dire qu'il différencie les majuscules des minuscules. Or, cela revêt son importance lors de l'utilisation de variables et d'objets.

Concrètement en JavaScript, `Monobjet` n'est pas identique à `monobjet`.

Cela s'applique à tous les mots clés (propriétés, méthodes, fonctions JavaScript) et l'usage d'outils de développement tel Aptana ou l'éditeur de Dreamweaver permet de faciliter la reconnaissance de cette syntaxe car ils sont transformés en couleurs quasi-instantanément.

Une autre règle de syntaxe concerne l'ajout de commentaires.

2. Ajout de commentaires

Comme dans la plupart des langages de programmation, l'ajout de commentaires dans vos scripts JavaScript peut s'avérer fort utile. En effet, outre le fait de pouvoir retrouver plus facilement les blocs d'instructions que vous venez de créer, les commentaires pourront vous être d'un immense secours le jour où vous devrez reprendre votre code. La lisibilité du code est, d'ailleurs, un des principaux critères de détermination d'un bon script JavaScript. En effet, à quoi sert de développer un superbe script si vous devrez prendre deux fois plus de temps pour le modifier quelques semaines plus tard ?

L'ajout de commentaires dans un bloc de code JavaScript s'effectue de manière monoligne ou multilignes.

Les commentaires contenus sur une seule ligne seront précédés du double slash //.

Les commentaires ne pouvant pas être contenus sur une seule ligne seront précédés de /* et devront se terminer par */.

Exemple :

```
<script language="javascript">
//Ceci est un commentaire monoligne
</script>
<script language="javascript">
/* Ceci est un commentaire
Sur plusieurs lignes */
</script>
```

Avec certains éditeurs HTML ou outil de conception, les commentaires apparaissent avec une couleur différente de celle du code.

3. Le point-virgule

Chaque ligne de code JavaScript se termine généralement par un point-virgule sauf exception de syntaxe que nous détaillerons plus tard.

Un simple oubli de ce point-virgule peut vous faire perdre un temps précieux. La première étape du débogage consistera donc, à vérifier sa présence.

4. L'indentation

Lorsque les lignes de codes commencent à être nombreuses, il peut arriver que le développeur soit un peu perdu devant des sigles qu'il ne parvient plus à relier. Il est alors utile d'utiliser une règle de présentation des scripts qui consiste à décaler vers la droite des instructions se correspondant. C'est notamment le cas lors de tests ou boucles imbriquées.

Exemple : afficher dans plusieurs boîtes de dialogue, le résultat de deux variables utilisées comme compteur dans deux boucles imbriquées.




```
<script language="javascript">
var i,j=0;
for (i=0;i<2;i++) {
alert("voici la valeur de mon premier compteur i: "+i);
    for (j=0;j<2;j++) {
    alert("voici la valeur de mon deuxième compteur j: "+j);
    }
}</script>
```

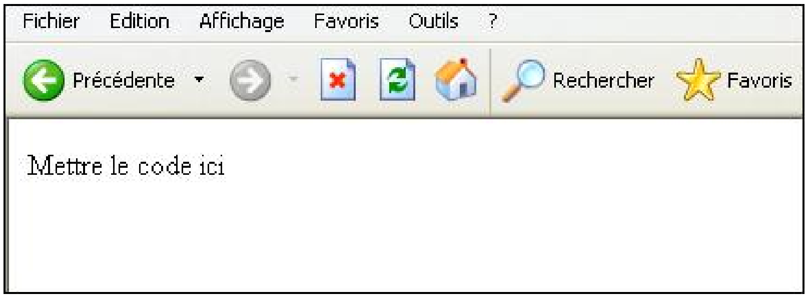
Ici le décalage des accolades par indentation permet de retrouver l'imbrication de la boucle j dans la boucle i. La création des boucles imbriquées sera détaillée dans le chapitre Contrôler les scripts avec les structures de contrôles.

La connaissance de ces éléments va vous permettre maintenant de rédiger la première page en JavaScript, qui vous servira également de page de référence.

Création de la page de test

Afin de rédiger plus efficacement les scripts dans les pages HTML, le mieux est de créer une page modèle dans laquelle vous inclurez les balises signalant le début et la fin du code JavaScript.

Le code JavaScript s'insérant le plus souvent dans la partie head de la page, voici le code HTML de votre page modèle pour la rédaction de tous vos scripts JavaScript :




```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Page modele en Javascript</title>
<script language="javascript">
document.write("Mettre le code ici");
</script>
</head>
<body>
</body>
</html>
```

Les deux premières lignes déterminent le type de document et sont nécessaires pour le bon fonctionnement d'instructions DHTML, comme nous le verrons au chapitre Améliorer l'interactivité avec JavaScript et CSS. La quatrième ligne indique le début de la balise head qui nous intéresse particulièrement. La cinquième ligne permet l'ajout d'une balise meta indiquant les caractères utilisés, la sixième fixe un titre à la page (ici Page modele en JavaScript). C'est entre les lignes <script language="javascript"> et </script> que devra s'écrire la plupart des scripts que vous allez rédiger. L'emplacement est repéré par la ligne document.write("Mettre le code ici").

Toutefois, il est possible de placer du code JavaScript n'importe où dans la page.

Enregistrez cette page modèle sous un nom permettant de la distinguer facilement (page modèle par exemple). Par la suite, vous pourrez :

- reprendre cette même page ;
- en modifier le script ;
- et enfin l'enregistrer sous un autre nom, à l'aide du menu **Enregistrer sous** de votre éditeur de texte ou HTML.

N'oubliez pas de modifier également le contenu de la balise <title>. Pour plus de sécurité et ne pas écraser cette page modèle, vous pouvez en modifier le droit d'accès et lui attribuer le statut lecture seule.

Exemple : pour créer une nouvelle page HTML comprenant un script d'écriture d'un message dans la page, il vous faut ouvrir la page modèle puis insérer entre les deux balises <script language="javascript"> et </script> le code suivant :

```
document.write(" la page modèle est ré-utilisable") ;
```

Avec cette méthode, vous vous constituerez facilement et rapidement une vraie bibliothèque de scripts.

Création d’une bibliothèque personnelle de scripts JavaScript

Au fil du temps, vous serez amené à développer de nombreux scripts qui seront peut-être réutilisables ultérieurement. Pour faciliter cette ré-exploitation, nommez vos pages avec un nom indiquant clairement l’objet de votre script JavaScript.

Attention à ne pas confondre bibliothèque personnelle de scripts et bibliothèques communes, nombreuses sur le net, et qui enrichissent le fonctionnement classique de JavaScript. L’installation de nouvelles bibliothèques JavaScript sera évoquée au chapitre Améliorer l’interactivité avec JavaScript et CSS.

Messages d’erreur et conseils pour le débogage

Le fait que les navigateurs interprètent différemment les scripts JavaScript, impose de faire des tests avec chacun des navigateurs. Cependant, le meilleur conseil est de tester les scripts d’abord dans Firefox/Mozilla qui dispose d’un outil de débogage plus puissant, puis, une vérification dans Internet Explorer sera nécessaire. Pour vous aider un peu dans le traitement des erreurs possibles, voici une typologie sommaire en trois grandes catégories :

Tout d’abord, il se peut que rien ne se passe au chargement de la page. Il faut savoir que JavaScript effectue un contrôle du script avant d’afficher quoique ce soit. S’il rencontre une erreur, le script s’interrompt sans même aller plus loin. Ces erreurs sont souvent dues à une syntaxe approximative ou à des fautes de frappes.

Il est possible également de rencontrer non pas des erreurs au chargement mais plutôt à l’exécution. Cela signifie généralement que les objets, leurs propriétés, leurs méthodes ou encore les fonctions ne correspondent pas ou sont mal utilisés.

Enfin, les erreurs les plus délicates à détecter sont des erreurs purement logiques qui surviennent lorsque les tests du script n’ont pas été suffisamment nombreux. Ainsi, le script peut tout à fait bien fonctionner dans un cas de figure et déclencher une erreur avec d’autres valeurs ou d’autres choix. N’hésitez pas à tester tous vos scripts avec des valeurs différentes et observez bien les résultats obtenus. Par prudence, s’il ne dispose pas d’un outil permettant de contrôler leur état (comme Aptana ou Firebug par exemple), le développeur devrait parsemer son script de nombreuses boîtes de dialogue. Celles-ci permettront d’afficher le contenu des variables tout au long du déroulement du script. D’un point de vue général, lorsqu’une erreur survient dans le déroulement de votre script, il est utile de chercher dans la barre d’état de son navigateur la présence d’une éventuelle icône signifiant que l’erreur a été repérée (sur Internet Explorer en bas à gauche, en bas à droite sur Firefox/Mozilla). En cliquant sur celle-ci, il vous sera possible de connaître la ligne du script contenant l’erreur. Mais attention, cette information n’est pas forcément précise. L’indication du numéro de ligne informe simplement qu’une erreur a été rencontrée à partir de cette ligne. Il faut donc remonter quelquefois plusieurs lignes plus haut, avant de trouver le problème à l’origine de l’arrêt du script. Une nouvelle fois, toutes les astuces (colorisation du code, indentation) sont les bienvenues, car ce travail, long et fastidieux, reste la principale cause du désintéressement de JavaScript.

Ces conseils constituent une première aide au débogage mais ils ne suffisent pas lorsque les problèmes s’avèrent plus délicat. Dans ce cas, il existe des outils permettant de faciliter un peu ce travail.

Les outils de débogage JavaScript

Malgré le fait qu'il soit possible de rédiger des scripts facilement, l'utilisation de certains outils de conception peut s'avérer utile notamment lors de l'étape de débogage. En effet, il sera possible d'utiliser alors des points d'arrêts, de connaître la valeur des variables, autant d'aides qui vous seront utiles dans cette étape cruciale qu'est la recherche des erreurs.

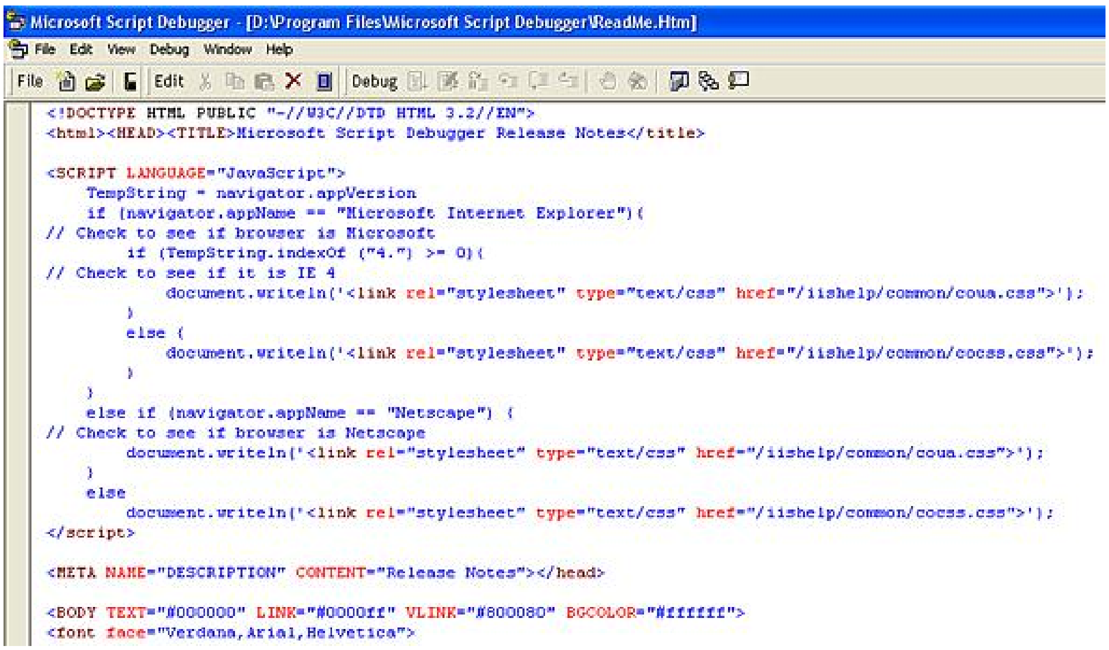
Nous allons présenter quelques-uns de ces outils.

1. Microsoft Script Debugger

En utilisant Microsoft Script Debugger, vous disposez d'un outil d'aide à la syntaxe et de débogage car lorsque le navigateur Internet Explorer rencontre une erreur dans le déroulement d'un script, il est possible de basculer directement dans Microsoft Script Debugger au niveau de la ligne posant un problème.

Il s'agit du logiciel de débogage de Microsoft mais son utilisation n'est pas très intuitive et il n'est disponible qu'en anglais. Vous pouvez tout de même le télécharger auprès du centre de téléchargement de Microsoft à l'adresse suivante : http://www.microsoft.com/downloads/

Vous aurez, peut-être, besoin d'installer auparavant le logiciel Genuine de Microsoft.

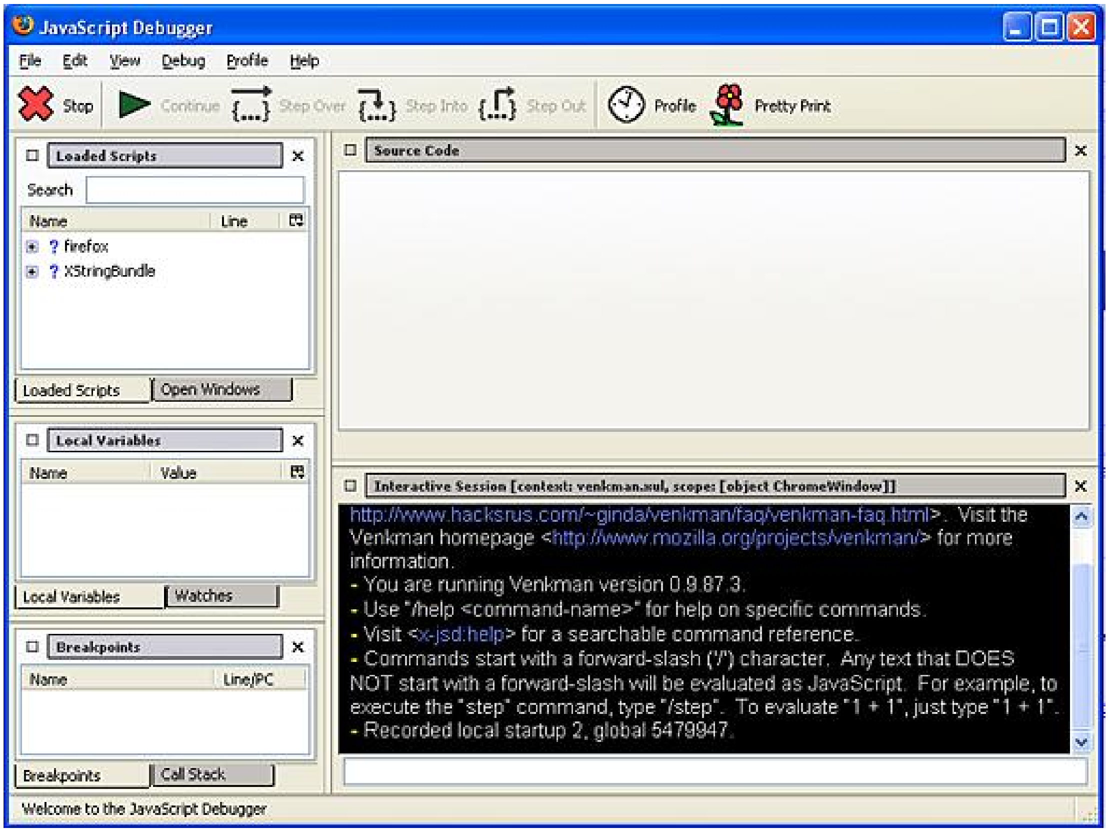


2. Microsoft FrontPage / Adobe Dreamweaver

Les éditeurs de code HTML permettent, en général, de visualiser le code de la page et ainsi d'accéder à la partie JavaScript. Mais ils ne disposent pas de fonctionnalités permettant d'ajouter des points d'arrêts et de connaître la valeur des variables. De ce fait, ils conviennent moins au débogage de script complexes mais apparaissent comme un bon pont de passage entre le monde HTML et le JavaScript.

3. Venkman

C'est un complément de Firefox/Mozilla (et s'utilise donc avec lui), qui permet de visualiser le code JavaScript, d'ajouter des points d'arrêts et de contrôler la valeur des variables. Il est intégré directement au navigateur et peut être lancé par le menu **Outils - JavaScript Debugger**.



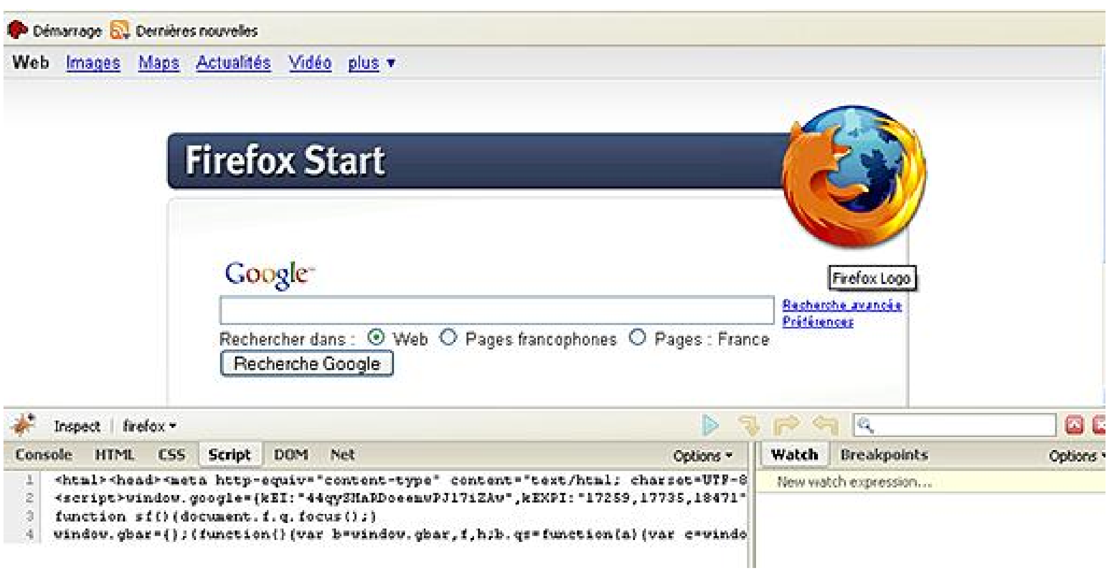
4. Firebug

Voici un autre complément de Firefox/Mozilla permettant les mêmes fonctionnalités que Venkman mais dont l'approche est peut-être plus intuitive. C'est cet outil que je vous conseille d'installer. Vous pourrez le trouver facilement en le téléchargeant à l'adresse suivante :

https://addons.Firefox/Mozilla.org/fr/Firefox/Firefox/Mozilla/addon/1843

Une fois le fichier téléchargé, il reste à l'installer en ouvrant le fichier par le menu **Fichier - Ouvrir un fichier...** puis à redémarrer Firefox/Firefox/Mozilla.

Firebug sera alors disponible par le menu **Outils - Firebug**.



Il est facile de constater que les outils sont plus nombreux avec Firefox/Mozilla. Qui plus est, les extensions sont nombreuses, gratuites et téléchargeables à l'adresse : https://addons.Firefox/Mozilla.org/fr/firefox/

C'est pour cela que Firefox/Mozilla reste le navigateur sur lequel il est plus facile d'effectuer un débogage, de

retrouver des variables, leur valeur ainsi que les objets sur lesquels la programmation JavaScript s'appuie.

Une fois votre outil sélectionné et votre page modèle créée, il est possible d'aborder les principaux concepts de la programmation orientée objet.

Langage de script et langage de programmation

Langage de script et langage de programmation sont souvent mis en opposition, pourtant, en les regardant de plus près, ils se ressemblent sur bien des points.

D'un côté, les langages de programmation nécessitent un éditeur spécifique et un compilateur. La rédaction d'un programme est également considérée comme plus complexe. Les instructions écrites par un langage de programmation sont interprétées directement par le processeur de la machine. Dans cette catégorie, nous retrouvons des langages tels que le C ou le Java.

D'un autre côté, les langages de script (dont fait partie JavaScript) permettent d'enchaîner une suite d'instructions qui sont exécutées par un autre programme (ici le navigateur Internet). Aucun compilateur n'est nécessaire et un seul éditeur de texte suffit. D'une manière générale, la rédaction de scripts est considérée comme plus aisée que la rédaction de programmes.

Pour simplifier, il est possible de dire que JavaScript fait appel à des programmes, comme les navigateurs (Internet Explorer, Firefox/Mozilla, etc.), pour exécuter une série d'instructions de manipulation d'objets (fenêtres, champs de page Internet, etc.) afin d'accomplir une application.

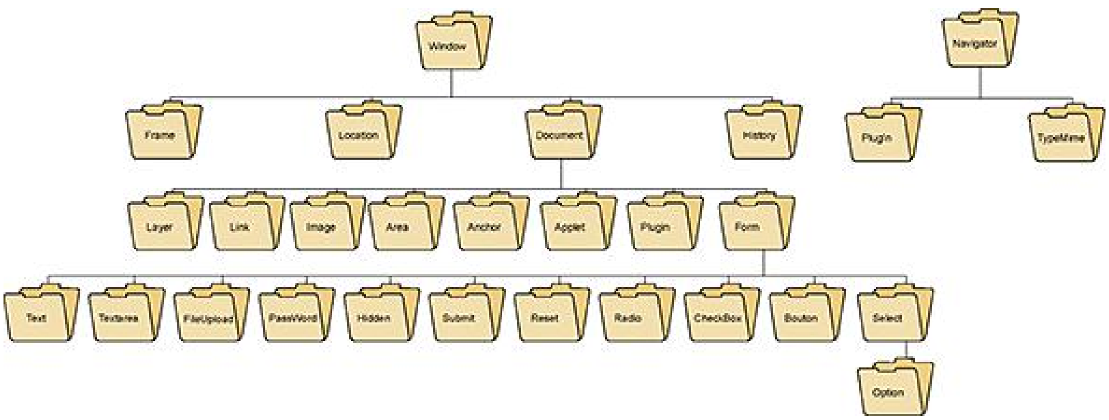
La manipulation de ces objets s'effectue selon une technique appelée Programmation Orientée Objet ou POO.

Les principes de la Programmation Orientée Objet

Ce type de programmation est né dans les années 60, puis a connu un fort développement à partir des années 80 notamment avec Smalltalk. Aujourd'hui, de nombreuses applications sont élaborées à partir de la programmation orientée objet qui, comme son nom l'indique, se base sur la notion d'objet.

1. Objets, Méthodes et Propriétés

La Programmation Orientée Objet est un paradigme informatique c'est-à-dire une façon de voir les choses. En fait, avec la Programmation Orientée Objet et notamment avec JavaScript, tous les éléments d'une page web (fenêtre, boutons, formulaire, champs de texte, etc.) sont considérés comme des objets que vous pouvez tester et manipuler. Mais ces objets n'ont pas tous la même importance. En effet, certains dépendent d'autres. Par exemple, un formulaire peut être composé de cases à cocher ou de champs texte qui sont donc, dépendants du formulaire, mais le formulaire lui-même, est dépendant du document dont il fait partie. Les termes notion de hiérarchie sont utilisés pour caractériser cette dépendance. Le diagramme suivant représente la hiérarchie des objets JavaScript prédéfinis :



Pour atteindre un objet, il est nécessaire de suivre le chemin (un peu comme dans un disque dur), en partant de l'objet le plus important hiérarchiquement pour, finalement, désigner celui qui l'est le moins. À chaque changement d'objet, un point est ajouté. Cette forme de syntaxe se nomme syntaxe pointée. Ainsi, dans une page comportant un formulaire nommé formu, contenant lui-même un champ Nom, si l'on désire accéder à ce champ, la syntaxe suivante sera utilisée :

```
document.formu.nom ;
```

Mais ce n'est pas tout, en plus d'atteindre un objet précis dans une hiérarchie, la syntaxe pointée désigne l'accès à des méthodes ou des propriétés qui permettent de manipuler ou de décrire ces objets. Ainsi, les propriétés constituent un ensemble d'attributs qui permettent d'en modifier l'apparence et les méthodes représentent des actions réalisables par cet objet. Pour expliquer plus concrètement la Programmation Orientée Objet, l'exemple d'une automobile est souvent pris. Selon cette métaphore, l'automobile représente un objet qui se caractérise par un certain nombre de propriétés (sa couleur bleue ou rouge, sa forme monospace ou berline). Parallèlement à ces propriétés, cet objet peut réaliser un certain nombre d'actions (avancer, tourner, reculer, etc.) qui correspondent à ce que l'on appelle des méthodes. Les méthodes et les propriétés ne sont pas forcément les mêmes pour tous les objets. Il arrive que plusieurs objets disposent de la même propriété ou de la même méthode, mais cela n'est pas obligatoire. En fait, pour bien utiliser JavaScript, il est nécessaire de bien connaître le modèle d'objet et les méthodes et propriétés disponibles.

D'un point de vue syntaxique, le point est utilisé pour séparer l'objet de sa propriété ou de sa méthode. Un des principaux objets prédéfinis de JavaScript est l'objet Window qui correspond à la fenêtre du navigateur.

Pour prendre un exemple concret, si vous désirez imprimer la page Internet active, vous utiliserez la méthode print() (imprimer) de l'objet window (fenêtre), qui est d'ailleurs l'objet le plus élevé dans la hiérarchie. Ainsi, le script JavaScript suivant permettra d'imprimer la page active :

```
<script language="javascript">
window.print();
</script>
```

L'objet window étant le plus élevé dans la hiérarchie, il n'est pas nécessaire de le nommer à chaque manipulation. Le script peut alors s'écrire encore plus simplement. Par la suite et par souci de simplification, nous ne ferons plus référence à l'objet window.

```
<script language="javascript">
print();
</script>
```

Autre exemple, vous désirez afficher dans la page la propriété title de l'objet document (qui permet d'obtenir le titre de la page), il est possible de le faire par le script suivant :

```
<script language="javascript">
document.write(document.title);
</script>
```

Ici, la méthode write() permet d'afficher, dans le document, la valeur correspondant à la propriété title de la page.

En JavaScript, il existe deux types d'objets, d'un côté les objets prédéfinis qui se retrouvent dans la hiérarchie d'objets dont nous avons déjà parlé et d'un autre côté, les objets créés par le développeur lui-même, lors de la création d'une fonction, comme nous le verrons dans le chapitre Fonctions et évènements. Pour ce qui est des objets prédéfinis, chaque nouvelle version de JavaScript enrichit le modèle en permettant d'accéder à de nouveaux objets ou en ajoutant de nouvelles méthodes et propriétés. Cet aspect plutôt positif entraîne inévitablement une autre conséquence, moins réjouissante. En effet, les visiteurs n'utilisant pas la dernière version du navigateur, ne pourront pas disposer des nouveaux objets, propriétés ou méthodes. Le script renverra immanquablement un message d'erreur. À chaque propriété ou méthode, correspondra alors une version de chaque navigateur !!! Un véritable casse-tête à première vue, mais avec l'explosion d'Internet (et la facilité des mises à jour), la grande majorité des internautes dispose aujourd'hui d'une version récente et adaptée. Au moment de la rédaction de cet ouvrage les versions 7.0 d'Internet Explorer et 2.0 de Firefox/Firefox/Mozilla supportent la version 1.5 de JavaScript.

Version de JavaScript	Navigateurs compatibles	Ajouts / Améliorations
1.0	Internet Explorer 3.0 Netscape Navigator 2.0	Version de base de JavaScript conformément à la directive de l'ECMA.
1.1	Internet Explorer 4.0 Netscape Navigator 3.0	Ajouts de nouveaux évènements. Correction de quelques bugs.
1.2	Internet Explorer 4.0 Netscape Navigator 4.0	Ajout de l'instruction switch. Ajout de propriétés de l'objet Navigator et de nouveaux évènements. Intégration des expressions régulières.
1.3	Netscape Navigator 4.5	Améliorations de l'objet Date. Correction de quelques bugs.
1.4	Netscape serveur	Développé uniquement pour les serveurs de Netscape. Ajouts des exceptions `jet` et `try ... catch` Ajout de nouveaux opérateurs (`instanceOf`). Changements apportés à LiveConnect. Modifications apportées à l'objet Array.
1.5	Netscape Navigator 6.0 Internet Explorer 6.0	Version basée sur les spécifications ECMA-262 3e édition. Ajout de la gestion des exceptions.
1.6	Firefox/Mozilla 1.0 Internet Explorer 6.0	Ajout de la technologie E4X permettant de gérer des documents XML. Ajout des méthodes every(), filter (), forEach(), map(), some() de l'objet Array.

1.7	Firefox/Mozilla 2.0 Internet Explorer 6.0	Ajout de nouveaux mots clés yield, let, de la définition des tableaux par compréhension.
1.8	Firefox/Mozilla 3.0	Ajout des fermetures d'expression, des expressions génératrices, des méthodes reduce() et reduceright() pour l'objet Array.

Pour utiliser une version spécifique de JavaScript (par exemple la version 1.7), il est plus prudent de le préciser lors de la déclaration de la balise script par l'instruction : <script language="javascript1.7">

La vraie difficulté réside plus dans le type du navigateur, car certaines méthodes ou propriétés peuvent ne pas être supportées par certains navigateurs et acceptées par d'autres. Un coup d'oeil jeté sur le tableau des méthodes et propriétés permet d'éviter toute perte de temps. C'est donc l'objet qui reste l'élément de base du langage JavaScript, et si vous avez besoin de faire appel à plusieurs méthodes ou propriétés pour un objet, plutôt que de citer plusieurs fois le même objet, vous pouvez utiliser le mot clé with dans l'instruction.

Il suffit de débuter l'instruction par le mot clé with, de mettre entre parenthèses l'objet puis d'utiliser les propriétés ou méthodes basées sur cet objet en les incluant entre des accolades. Tout ce qui se trouve entre ces accolades se rapporte à l'objet cité. La syntaxe est donc, la suivante :

```
with(nomdelobjet) {
Instructions de manipulation des propriétés ou méthodes ;
}
```

Exemple : afficher le contenu d'un champ de formulaire avant et après modification de la valeur de la variable.




```
<html>
<head>
<title>Utilisation de With</title>
<script language="javascript">
function controle() {
with (document.form1.nom) {
value="premiere valeur";
alert(value);
value="nouvelle valeur";
alert(value);
}
}
</script>
</head>
<body>
<form name="form1" method="post" action="">
  <p>
    <input name="nom" type="text" id="nom">
</p>
  <p>
```

```
    <input type="button" name="Submit" value="Bouton" onClick="controle()">
</p>
</form>
</body>
</html>
```

Ici, l’objet correspondant au champ texte de formulaire est, d’abord, déclaré par l’instruction with (document.form1.nom). Il est ensuite inutile de reprendre la syntaxe document.form1.nom, il suffit d’ajouter les instructions modifiant successivement la valeur de l’objet. L’affichage successif de plusieurs boîtes de dialogues permet de suivre cette modification de valeur.

Vous pourriez penser qu’une des conditions essentielles de l’apprentissage de la Programmation Orientée Objet est la connaissance parfaite de l’ensemble du modèle d’objets JavaScript. Pourtant, il est préférable, pour débuter, de connaître les propriétés et méthodes essentielles des objets qui sont manipulés le plus souvent.

Méthodes JavaScript

1. alert()

La méthode alert() de l'objet window permet l'affichage d'une boîte de dialogue comprenant un message d'avertissement. Le message à afficher dans la boîte de dialogue se trouvant entre guillemets, il est possible d'y écrire avec des accents, des espaces et tout autre caractère que vous souhaitez voir s'afficher. Vous pouvez aussi afficher des variables à la place ou en plus du message, comme nous le verrons au chapitre Utilisation des constantes, variables et opérateurs.

Lors de l'affichage de cette boîte de dialogue, JavaScript stoppe son déroulement et attend que l'utilisateur clique sur le seul bouton **OK**.

Cette méthode, même si elle semble limitée au premier abord, vous sera bien utile par la suite lorsqu'il s'agira d'effectuer un débogage du script et de vérifier la valeur des variables.

Exemple : afficher une boîte de dialogue de type alert avec un simple message.




```
<script language="javascript">
alert("Ceci est une boîte de dialogue affichée par JavaScript");
</script>
```

Si vous souhaitez afficher un message sur deux lignes, vous devez ajouter \n à la fin de la chaîne de caractères de la première ligne.

```
<script language="javascript">
alert("Ceci est une boîte de dialogue affichée \n sur 2 lignes");
</script>
```

Mais l'emploi de la méthode alert() reste limité, car l'utilisateur n'a pas de possibilité pour entrer des données et répondre ainsi au message qui est apparu. Pour cela, il faut utiliser d'autres méthodes fournies par JavaScript.

2. confirm()

La méthode confirm() de l'objet window affiche une boîte de dialogue avec un message suivi des deux boutons **OK** et **Annuler**.

Exemple : afficher une boîte de dialogue au chargement de la page avec deux boutons de réponse par la méthode Confirm().

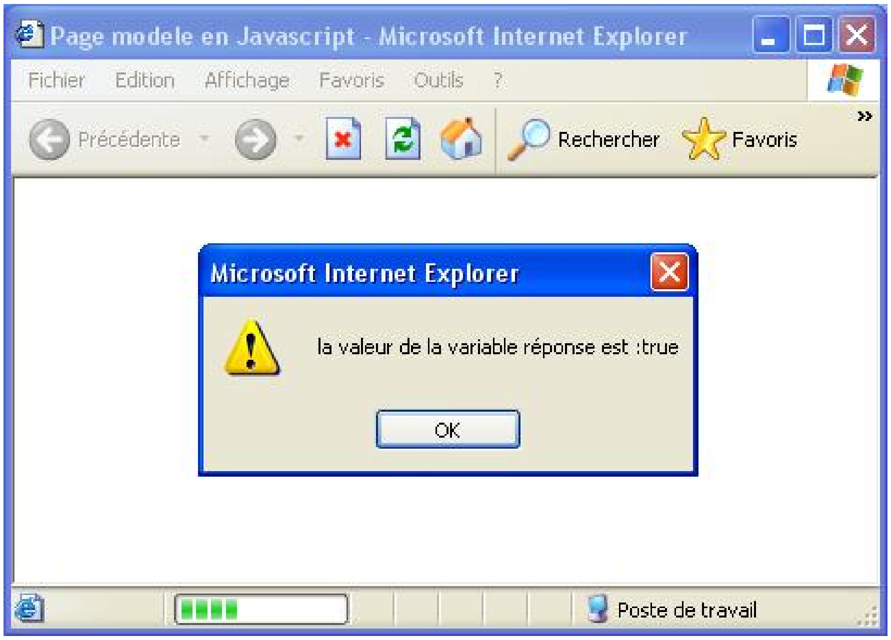



```
<script language="javascript">
confirm("ok = 0, Annuler = 1");
</script>
```

La grande différence avec la méthode alert() c'est qu'en fonction de la réponse, c'est-à-dire du clic sur **OK** ou **Annuler**, vous pouvez renvoyer une valeur (true pour **OK** et False pour **Annuler**) vers une variable. Il suffit de faire précéder la méthode confirm() par la déclaration de la variable réponse. L'utilisation des variables est détaillée au chapitre Utilisation des constantes, variables et opérateurs mais vous pouvez tout de même modifier le script précédent :

Exemple : afficher une boîte de dialogue avec deux boutons OK et Annuler, puis renvoyer la valeur de la question posée dans une autre boîte de dialogue.






```
<script language="javascript">
var reponse=window.confirm("ok = true, Annuler = false");
alert("la valeur de la variable réponse est : "+reponse);
</script>
```

Pour débuter, il faut déclarer une variable appelée, ici, reponse et lui affecter une valeur qui correspond au choix de l'utilisateur (c'est-à-dire à la valeur 0 ou -1). Ensuite, il suffit d'afficher la réponse à l'aide de la méthode alert() en prenant soin de mettre entre guillemets le message à faire apparaître, d'ajouter l'opérateur de concaténation + ainsi que la variable. Vous pouvez ainsi construire une véritable phrase comme réponse.

Même si la méthode confirm() permet un début d'interaction, la réponse ne peut correspondre qu'à deux valeurs au plus. Dans le cas d'une réponse ouverte, il faut passer par une troisième méthode.

3. prompt()

La méthode prompt() permet, en plus, d'afficher un message pour saisir une valeur dans un champ, appelé invite. Il est donc possible de saisir une réponse ouverte, contrairement aux deux précédentes méthodes. La boîte de dialogue dispose de deux boutons, le bouton **OK** permet d'affecter la valeur saisie dans l'invite, à une variable, le bouton **Annuler** entraîne l'affectation de la valeur `Null` à la variable. Il est aussi possible d'afficher une valeur par défaut dans le champ servant à recevoir la réponse, et si elle ne convient pas à l'utilisateur, celui-ci pourra saisir par-dessus.

La syntaxe de la méthode prompt() est la suivante :

```
Prompt("texte à afficher", "valeur par défaut") ;
```

Exemple : afficher deux boîtes de dialogue afin de saisir le nom et le prénom du visiteur puis en ouvrir une troisième affichant le nom complet.






```
<script language="javascript">
nom=prompt("Quel est votre nom ?", "Saisir votre nom ici");
prenom=prompt("Quel est votre prenom ?", "Saisir votre prenom ici");
alert("Votre nom complet est : \r"+nom+"\r "+prenom);
</script>
```

Dans un premier temps, l'utilisation de la méthode prompt() permet de demander le nom et le prénom au visiteur. Les valeurs saisies sont automatiquement affectées à deux variables nom et prenom. Ensuite, il est facile de construire le message affiché par la méthode alert(). Ce message sera l'addition d'une chaîne de texte (placée entre guillemets), et de variables qui s'afficheront sur deux lignes grâce à l'usage de '\r'.

Attention, le symbole '\r' n'est pas pris en charge par Firefox/Mozilla. Dans ce contexte, les réponses s'affichent sur la même ligne.

L'utilisation de ces méthodes de l'objet window est liée aux variables, dont il faut connaître le fonctionnement.

Typologie et utilisation des constantes

Les constantes sont des éléments d'informations, dont les valeurs sont indiquées explicitement dans le code JavaScript. En règle générale, les valeurs des constantes sont déterminées au début du script et sont valables jusqu'à la fin. Cette affectation de la valeur des constantes s'effectue selon la syntaxe suivante :

`Ma_constante` = valeur de la constante ;

La valeur de la constante inscrite à droite du signe égal est stockée dans la partie de la mémoire désignée à gauche du signe égal et nommée, dans cet exemple, Ma_constante.

Les constantes JavaScript peuvent être classées en plusieurs catégories :

1. Constantes arithmétiques

Elles correspondent à un nombre (`type number`), constitué d'une suite de chiffres. Il existe plusieurs types de constantes arithmétiques en fonction de leur mode d'écriture :

La constante décimale est constituée d'une suite de 17 chiffres maximum allant de zéro à neuf. La constante décimale peut être négative et donc, être précédée du signe moins.

Exemple :

64 ou -64 ou 64.14 ou 3.1415926535 sont des constantes décimales.

Les constantes décimales utilisent le point et non la virgule pour séparer la partie entière de la partie fractionnaire. Si vous utilisez la virgule, la partie fractionnaire n'est pas prise en compte.

La constante octale est constituée d'une suite de chiffres allant de zéro à sept. La constante octale débute par un zéro et doit être obligatoirement entière et positive.

Exemple :

02542571 est une constante octale.

La constante hexadécimale est constituée d'une suite de seize caractères comprenant des chiffres décimaux de zéro à neuf auxquels on ajoute des lettres de A à F. La constante hexadécimale doit commencer par un zéro suivi d'un x.

Exemple :

0xF45B est une constante hexadécimale.

2. Constantes chaînes de caractères

Elles correspondent à une suite de caractères qui peuvent être des lettres ou des chiffres ou une association des deux. La chaîne de caractères doit être encadrée par des guillemets ou des apostrophes.

Exemple :

"`ma constante texte`" ou '`ma constante texte`' sont des constantes chaînes de caractères.

Une variable texte peut être vide, dans ce cas on écrit deux apostrophes ou guillemets successifs sans rien à l'intérieur.

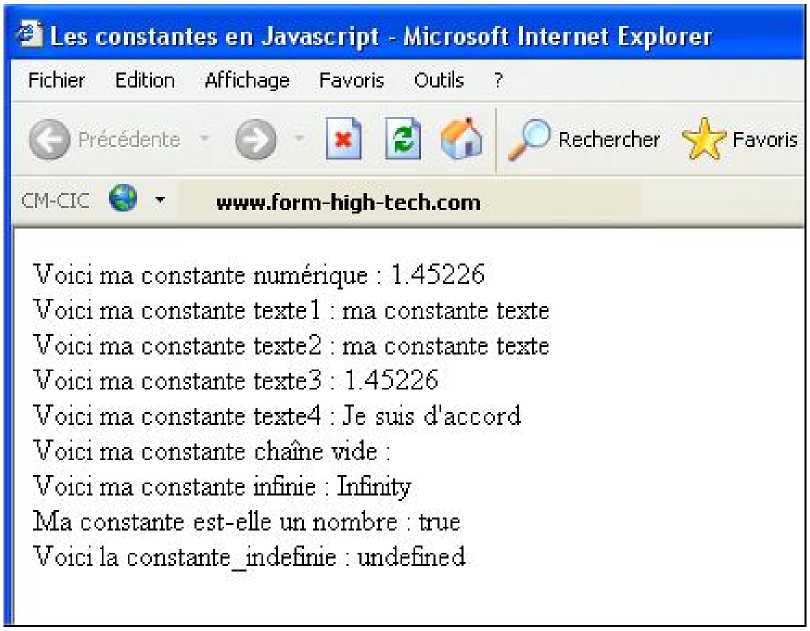
3. Constantes booléennes

Elles ne peuvent correspondre qu'à deux valeurs, true ou false, écrites sans guillemets ni apostrophes et établies, le plus généralement, en fonction d'un test dans des structures de contrôle, que nous détaillerons au chapitre Contrôler les scripts avec les structures de contrôles.

Exemple :

`Reponse = true` permet l'affectation de la valeur booléenne true à la constante Reponse.

Exemple général sur les constantes : déclarer un certain nombre de constantes en fonction du type et de la syntaxe, puis afficher les résultats dans la page par la méthode document.write().




```
<script language="javascript">
constante_numerique=1.45226;
constante_texte1="ma constante texte";
constante_texte2='ma constante texte';
constante_texte3="1.45226";
constante_texte4="Je suis d\'accord";
constante_chaine_vide=""
constante_infinie=7.6E+333*6.7E+333;
constante_nombre="ceci n'est pas un nombre";
constante_indefinie= undefined;
document.write ("Voici ma constante numérique :
"+constante_numerique+"<BR>");
document.write ("Voici ma constante texte1 :
"+constante_texte1+"<BR>");
document.write ("Voici ma constante texte2 :
"+constante_texte2+"<BR>");
document.write ("Voici ma constante texte3 :
"+constante_texte3+"<BR>");
document.write ("Voici ma constante texte4 :
"+constante_texte4+"<BR>");
document.write ("Voici ma constante chaîne vide :
"+constante_chaine_vide+"<BR>");
document.write ("Voici ma constante infinie :
"+constante_infinie+"<BR>");
document.write ("Ma constante est-elle un nombre :
"+isNaN(constante_nombre)+"<BR>");
document.write ("Voici la constante_indefinie :
"+constante_indefinie);
</script>
```

Il suffit simplement, ici, d'effectuer des déclarations et des affectations de constantes puis d'en afficher les valeurs par l'intermédiaire de la méthode document.write(). Celle-ci permet d'écrire directement dans la page et l'utilisation de
 permet de faire un retour à la ligne afin de clarifier l'affichage. Il faut noter, tout de même, l'utilisation du mot clé undefined correspondant à une constante JavaScript.

4. Autres constantes

En plus des constantes définies par l'utilisateur, l'ECMA a défini quelques constantes :

`Infinity`

Cette constante correspond à un nombre infini, supérieur à $1.7976931348623157E+10^{308}$ ou inférieur à $-1.7976931348623157E+10^{308}$.

Cette constante est renvoyée comme résultat à une division comportant un diviseur égal à zéro.

Exemple :

```
<script language="javascript">
a=5; b=0;
resultat=a/b;
alert(resultat);
</script>
```

`NaN`ou `isNaN`

Cette constante qui signifie (Not a Number) permet de vérifier si la variable est bien un nombre. Il faut utiliser la syntaxe suivante avec `isNaN` :

```
isNaN(variable ou constante) ;
```

`IsNaN` renvoie true si la valeur testée n'est pas un nombre et false si cela en est un.

`Undefined`

Cette constante prend la valeur true dans deux types de situations. Le premier correspond au cas où une variable n'est pas déclarée, le second correspond au cas où une variable est bien déclarée mais n'a pas de valeur affectée.

Mais `undefined` correspond également à un type de variable. Il est fréquent de l'utiliser avec le mot clé typeof dans un test pour connaître l'état de la variable.

Même si leur utilité ne peut être remise en cause, les constantes sont beaucoup moins employées que les variables, dont la manipulation est systématique.

Typologie des variables

Contrairement aux constantes dont la valeur reste fixe tout au long du script, les variables peuvent changer de valeur soit par une nouvelle affectation directe, soit par un calcul dans le code ou encore par une affectation par la méthode prompt(). Une variable peut, d'ailleurs, changer de valeur tout au long du déroulement du script puisque JavaScript retient la dernière définition de la variable, c'est là que réside l'intérêt de leur utilisation. Inversement, si la variable ne doit pas changer de valeur durant tout le déroulement du script, il est possible de la transformer en constante à l'aide du mot clé const. Autre différence mais cette fois-ci avec les autres langages de programmation, JavaScript est faiblement typé c'est-à-dire que les variables n'ont pas besoin de correspondre à un type (texte, numérique, booléen) pour fonctionner. En fait, c'est l'interpréteur JavaScript qui définit le type de variable au moment de son exécution. Cela implique un avantage et un inconvénient. L'avantage réside dans le fait qu'en se passant de la détermination du type, vous économisez du code, l'inconvénient c'est que, parfois, vous pouvez aboutir à des incohérences : par exemple, additionner du texte et des nombres sans que cela gêne JavaScript. Pour conclure, les types de variables correspondent à ceux des constantes (texte, numérique ou booléen), et il est donc, simple de les utiliser.

Les étapes à respecter pour une bonne utilisation des variables

Afin de bien manipuler les variables, il est utile d'en respecter les étapes de création. En règle générale, il faut commencer par se préoccuper de la portée des variables pour ensuite effectuer leur déclaration et leur affectation.

1. Portée des variables

Il faut se poser la question de savoir si la variable doit être disponible tout au long du script, ou si elle doit être limitée à la fonction dans laquelle elle a été créée. C'est ce que l'on appelle la portée des variables.

Il existe deux types de portées différentes. Lorsqu'une variable est définie à l'intérieur d'une fonction, nous le verrons dans le chapitre consacré aux fonctions, sa portée est dite locale. Dans ce cas, sa valeur ne peut pas être récupérée dans une autre fonction, sans en faire explicitement référence. Par contre, lorsqu'une variable est définie directement dans le script et sans appartenir à aucune fonction, sa portée est dite globale. Sa valeur est, alors, disponible à tout moment.

2. Déclaration de variables

La déclaration d'une variable s'effectue à l'aide du mot clé var et selon la syntaxe suivante, on parle alors de déclaration explicite :

```
var nom_de_la_variable ;
```

Il est possible de déclarer plusieurs variables sur une seule ligne. Dans ce cas, elles seront séparées par une virgule.

```
var mavariable1, mavariable2, mavariable3 ;
```

Dans ce cas de figure, les variables n'ayant pas reçu de valeurs, elles sont dites `undefined`. Aussi, il est important de voir comment vous pouvez réaliser l'affectation des variables.

Mais, JavaScript est un langage particulièrement souple et il autorise, aussi, la déclaration des variables sans utiliser le mot clé var, cela s'appelle une déclaration implicite.

3. Affectation de variables

En JavaScript, l'affectation peut s'effectuer selon trois manières différentes. Il est possible de le faire directement dans le code par l'usage du signe égal (comme pour les constantes, la valeur située à droite du signe égal est affectée à la variable dont le nom figure à gauche du signe égal).

```
var mavariable = 10 ;
```

Il est possible de le faire indirectement par l'intermédiaire d'un calcul impliquant constantes ou variables.

```
var somme=mavariable1+mavariable2 ;
```

Enfin, il est possible de demander à l'utilisateur une valeur qui sera affectée à la variable par la méthode prompt(), vue précédemment.

```
var reponse=prompt(" Quelle est la valeur de cette variable ? ") ;
```

Ce type de syntaxe même si elle respecte à la lettre la bonne procédure de programmation n'est pas la seule. En effet, ce type de déclaration dit « explicite » n'est pas obligatoire et il est possible de faire une déclaration implicite des variables sans utiliser le mot clé var.

La syntaxe devient alors :

```
mavariable = 10 ;
```

ou encore :

```
somme=mavariable1+mavariable2 ;
```

Une fois les variables affectées de leurs valeurs, il est possible de les afficher.

4. Affichage des variables

L'affichage des variables s'effectue, généralement, par l'intermédiaire des méthodes alert() ou confirm(), détaillées au chapitre Initiation à la Programmation Orientée Objet et JavaScript.

Exemple 1 : afficher dans une boîte de dialogue le résultat d'une addition de deux variables :

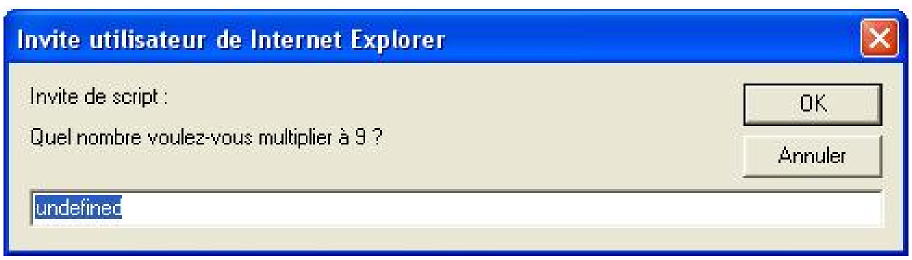



```
<script language="javascript">
var somme, nombre1=10;nombre2=5;
somme=nombre1+nombre2;
alert("le résultat de l'addition est : "+somme);
</script>
```

Tout d'abord, la première instruction permet de déclarer trois variables (deux pour chacun des nombres et une pour le résultat) pour permettre la réalisation du calcul. Évidemment, il est possible d'ajouter autant de variables que l'on désire par un calcul plus complexe. L'instruction suivante permet d'effectuer le calcul. Ensuite, le résultat de l'opération est affiché après une chaîne de texte de présentation.

Le symbole + est, dans ce cas, un opérateur de concaténation de chaînes et non pas un opérateur arithmétique, comme c'est le cas dans la ligne précédente.

Exemple 2 : afficher le résultat d'une multiplication dans la page entre une variable égale à neuf et une autre variable, dont la valeur sera saisie dans une boîte de dialogue par l'utilisateur.




```
<script language="javascript">
var nombre1=9;
var nombre2=prompt("Quel nombre voulez-vous multiplier à 9 ?");
var produit=nombre1*nombre2;
document.write("le résultat de la multiplication est : "+produit);
</script>
```

Dans un premier temps, il faut affecter la variable nombre1 avec la valeur 9. Puis, il faut demander à l'utilisateur de saisir une valeur qui est affectée à la variable nombre2. Tant que l'utilisateur n'a rien saisi, la variable nombre2 est undefined. Ensuite, le script effectue le produit des deux variables. Enfin, le résultat est affiché.

Exemple 3 : afficher le résultat d'un calcul de coût d'un trajet pour un véhicule en fonction de la consommation moyenne. L'utilisateur doit renseigner plusieurs boîtes de dialogue afin d'obtenir la marque, la consommation moyenne pour cent kilomètres, le coût du litre de carburant utilisé et le nombre de kilomètres effectués.






```
<script language="javascript">
var marque=prompt("Quelle est la marque de votre véhicule ?",
" saisissez ici la marque de votre véhicule");
var consommation=prompt("Quelle est la consommation en litres de
votre véhicule pour 100 kilomètres ?", "saisissez ici la
consommation en litres de votre véhicule");
var prix_litre=prompt("Quel est le prix en euros du litre de votre
carburant ?", "saisissez ici le prix du litre de carburant");
var nb_kilometres=prompt("Quel est le nombre de kilomètres de votre
parcours ?", "saisissez ici le nombre de kilometres de votre parcours");
var cout_total=nb_kilometres/100*consommation*prix_litre;
alert("Le cout total d'un parcours de " + nb_kilometres +
" kilomètres effectué avec un véhicule de marque " + marque + "
consommant " + consommation + " litres pour 100 kilomètres en moyenne,
avec un prix au litre du carburant de " + prix_litre + " euros
est de " + cout_total + " euros");
</script>
```

Il s'agit tout d'abord de demander à l'utilisateur de renseigner toutes les variables nécessaires par l'intermédiaire de la méthode prompt(). Les variables marque, consommation, prix_litre et nb_kilomètres sont ainsi affectées. Avec toutes ces informations, il faut calculer le coût total du trajet en divisant le nombre de kilomètres par 100 et en multipliant le résultat, par la consommation moyenne et le prix au litre. Pour terminer, il est plus agréable d'afficher le résultat final dans une boîte de type alert() en alternant les affichages entre les variables et les chaînes de caractères.

5. Transfert de valeurs entre variables et conversion de type

Pour donner la valeur d'une variable à une autre, il suffit d'utiliser le signe égal pour transférer la valeur de la variable à droite du signe égal dans celle située à gauche du signe égal.

```
var mavariable2=mavariable1 ;
```

Il est possible de convertir une variable d'un type à un autre (généralement de texte vers du numérique) grâce aux méthodes parseInt() (conversion de texte vers un nombre entier) et parseFloat() (conversion de texte vers un nombre décimal). La syntaxe est la suivante :

```
var variablenumerique=parseInt(variabletexte) ;
```

```
var variablenumerique=parseFloat(variabletexte) ;
```

Exemple : afficher la valeur d'une variable avant et après conversion par la méthode parseInt().






```
<script language="javascript">
var nombre=15;
var nombreentexte="10";
var nombreenchiffre=parseInt(nombreentexte);
somme=nombre+nombreentexte;
alert("Voici le résultat avant conversion : "+somme);
somme=nombre+nombreenchiffre;
alert("Voici le résultat après conversion : "+somme);
</script>
```

Comme d’habitude, les variables sont déclarées et affectées dès le début du script. La variable nombre correspond au nombre 15, tandis que la variable nombreentexte est affectée de la chaîne de caractères 10. Il n’est, donc, pas possible d’effectuer un calcul avec ces deux variables. La variable nombreenchiffre déclarée ensuite, correspond à la conversion en chiffre de la variable nombreentexte. Pour être certain de la réussite de la conversion, il est possible d’afficher tout d’abord le résultat de la somme avant conversion correspondant à nombre+nombreentexte (le résultat correspond à la concaténation des deux chaînes de caractères : 1510), puis d’afficher le résultat après conversion nombre+nombreenchiffre (le résultat est de 25, prouvant que la conversion s’est bien déroulée).

Dans le cas où la conversion ne peut s’effectuer, la valeur NaN (is Not a Number, ce qui signifie n’est pas un nombre) sera renvoyée.

À l’inverse, il est possible de convertir une variable numérique en texte par l’intermédiaire de la méthode toString() dont la syntaxe est la suivante :

```
Var variabletexte=variablenumerique.toString() ;
```

<u>*Exemple : afficher le résultat de la conversion de nombre en texte d’une variable.*</u>




```
<script language="javascript">
var nombreenchiffre=15;
var nombreentexte=nombreenchiffre.toString();
total = 10+nombreentexte;
alert("Le total est :" + total);
</script>
```

Le début correspond à l’affectation du chiffre 15 à la variable nombreenchiffre. Ensuite, il faut appliquer la méthode toString() à la variable pour la convertir en texte dans nombreentexte. Après conversion, l’addition entre la variable numérique et la variable convertie renvoie une concaténation de variables textes, démontrant ainsi que la conversion s’est bien déroulée.

Règles de nommage et mots réservés

Jusqu'à présent, nous avons utilisé des noms de variables très conventionnels et relativement explicites, mais au fur et à mesure de votre progression, vous serez peut-être tenté d'utiliser des noms de variables plus personnalisés. Cependant, il est impératif de suivre certaines contraintes et de respecter certaines règles, sous peine de voir le script renvoyer une erreur.

Ainsi, les noms des variables ne peuvent pas contenir d'espaces. En général, il faut utiliser l'underscore « _ » pour symboliser un espace.

De même, le premier caractère d'un nom de variable ne peut pas être un point.

Enfin, les noms des variables peuvent contenir, indifféremment, des majuscules ou des minuscules mais il est important de bien respecter ce changement de casse, car JavaScript y est sensible. Cela signifie que Mavariable et mavariable sont, pour JavaScript, deux variables différentes.

1. Conseils pour le nommage

Il est préférable de nommer les variables explicitement, c'est-à-dire pour ce qu'elles représentent plutôt que de prendre des noms de variables totalement abstraites. Ainsi, vous pourrez les retrouver plus facilement dans le code lors du débogage. Il existe, cependant, une liste de mots qu'il est totalement interdit d'utiliser lors de la déclaration de vos variables, car réservés par JavaScript.

2. Mots réservés

Les mots réservés sont des mots correspondant généralement à des objets, propriétés ou méthodes déjà utilisés par JavaScript et qui ne peuvent, donc, pas recevoir de valeurs. Voici un tableau qui récapitule ces mots interdits. Certains de ces mots sont d'ores et déjà interdits, d'autres, même s'il est encore possible de les utiliser, sont déconseillés car les prochaines versions de JavaScript ne les accepteront plus.

abstract	float	short
boolean	for	static
break	function	super
byte	goto	swich
case	if	synchronized
char	implements	this
class	import	throw
const	in	throws
continue	instanceoff	transient
debugger	int	true
default	interface	try
delete	long	typeoff
do	native	var
double	new	void
else	null	volatile

enum	package	while
export	private	with
extends	protected	
false	public	
finally	return	

Typologie des opérateurs

Les opérateurs permettent de manipuler les variables, de faire des tests (comparaison) entre les valeurs des variables. Les opérateurs et les variables composent une expression qui peut, quelquefois, ressembler à un message crypté. Mais, en les étudiant, la maîtrise des opérateurs est somme toute relativement simple. Nous découvrirons au chapitre Contrôler les scripts avec les structures de contrôle, comment utiliser ces tests dans le déroulement du script. Pour l'instant, voici la liste des différents opérateurs et leur signification :

1. Arithmétiques

Les opérateurs arithmétiques permettent de réaliser des calculs élémentaires entre des variables de types numériques. Les symboles +, -, * et / permettent d'effectuer les opérations élémentaires d'addition, de soustraction, de multiplication et de division. Voici un tableau qui liste les opérateurs arithmétiques :

Opérateurs	Nom	Rôle	Exemple	Résultat
+	Plus	Additionne les valeurs situées à gauche et à droite du symbole.	15+10	25
-	Moins	Soustrait les valeurs situées à gauche et à droite du symbole.	15-10	5
*	Produit ou multiplié	Effectue la multiplication entre les valeurs situées à gauche et à droite du symbole.	15*10	150
/	Divisé	Effectue la division de la valeur située à gauche par la valeur à droite du symbole.	15/10	1.5
%	Modulo	Extrait la valeur entière du résultat de la division entre la valeur située à gauche et la valeur située à droite du symbole modulo.	5%3	1
++	incrémentation	Permet d'incrémenter une valeur notamment dans le cas d'utilisation dans une boucle.	Variable=1 Variable++	2
--	décrémentation	Permet de décrémenter une valeur notamment dans le cas d'utilisation dans une boucle.	Variable=2 Variable--	1
-	Négation	Permet d'obtenir l'opposé.	Variable=1 variable=-variable	-1

Petite précision pour le symbole modulo « % », qui permet de récupérer le reste de la division de l'opérande de gauche par celui de droite. En affectant le symbole modulo à une variable, vous obtenez le reste de la division de la variable par le chiffre situé à droite du symbole. En voici la syntaxe : `modulo=dividende%diviseur`.

Exemple : afficher le résultat de l'application de modulo trois sur une variable égale à cinq.

```
<script language="javascript">
var mavariable=5;
var modulo = mavariable%3;
alert("Le resultat de modulo est : " + modulo);
</script>
```

Après avoir affectée cinq à la variable, celle-ci est réutilisée dans le calcul d'une autre variable appelée modulo. Enfin, le résultat est affiché dans une boîte de dialogue par la méthode alert().

Les opérateurs d'incrémentation et de décrémentation peuvent être positionnés avant l'utilisation de la variable, il s'agit de pré-incrémentation (ou pré-décrémentation), ou être positionnés après l'utilisation de la variable et il s'agit alors de post-incrémentation (ou post-décrémentation). L'utilisation de ces opérateurs sera détaillée lors de l'étude des boucles au chapitre suivant.

2. Comparaison

Ces opérateurs permettent de comparer des variables entre elles.

Opérateurs	Nom	Rôle	Exemple	Résultat
<	Inférieur	Teste si l'opérande à gauche du symbole est plus petit que celui situé à droite. Renvoie true si c'est le cas.	1<2	True
<=	Inférieur ou égal	Teste si l'opérande à gauche du symbole est plus petit ou égal à celui situé à droite. Renvoie true si c'est le cas.	2<=2	True
==	Egal	Teste si les valeurs à gauche et à droite sont identiques.	2==2	True
>	Supérieur	Teste si l'opérande à gauche du symbole est plus grand que celui situé à droite. Renvoie true si c'est le cas.	2>1	True
>=	Supérieur ou égal	Teste si la valeur à gauche du symbole est plus grande ou égale à celle située à droite. Renvoie true si c'est le cas.	2>=2	True
!=	Différent	Teste si l'opérande situé à gauche du symbole est différent de celui de droite. Renvoie true si c'est le cas.	1 !=2	True

===	Strictement égal	Teste si l'opérande situé à gauche est égal et du même type que celui situé à droite de l'opérateur.	1=== "1 "	False
!==	Strictement différent	Teste si l'opérande situé à gauche est différent et/ou de type différent de celui situé à droite de l'opérateur.	"1 " !== 1	True

Ne pas confondre l'opérateur de comparaison == avec l'opérateur d'affectation =, ce dernier sert à affecter une valeur dans la variable située à gauche du symbole.

Exemple : afficher la valeur correspondant à la réponse true ou false de JavaScript après exécution d'un test de comparaison.

```
<script language="javascript">
var reponse=5>12;
alert("Le résultat du test est : " + reponse);
</script>
```

Le script se limite à un test 5>12 dont la réponse est renvoyée dans un variable nommée reponse, dont le contenu sera affiché pour connaître le résultat du test. Le résultat de ce test est exprimé de façon booléenne (c'est-à-dire true ou false).

Il est possible d'utiliser le même script en modifiant l'opérateur de comparaison pour appréhender leur mode de fonctionnement et visualiser le résultat renvoyé.

Par exemple avec l'opérateur != (différent), le résultat est true.

```
<script language="javascript">
var reponse=5 !=12;
alert("Le résultat du test est : " + reponse);
</script>
```

3. Logiques

Ils sont aussi appelés les opérateurs booléens. Utilisés avec plusieurs tests, ils renvoient une valeur true ou false en fonction du respect de la ou des conditions. Les opérateurs && et || sont dits binaires, alors que l'opérateur ! est dit unaire.

Opérateurs	Nom	Rôle	Exemple	Résultat
&&	Et	Permet de concaténer plusieurs conditions en vue d'un test. Renvoie true si les deux conditions sont réunies.	5 est supérieur à 4 et inférieur à 6	True
||	Ou	Vérifie que l'une ou l'autre des conditions est remplie. Renvoie true si c'est le cas.	5 est supérieur à 4 ou inférieur à 3	True
!	Non	Vérifie que la condition du test n'est pas	5 est supérieur à 6	True

		respectée. Renvoie true si c'est le cas.		

<u>Exemple 1 : afficher le résultat d'un double test (5<12 et 5<3) où les deux conditions doivent être respectées :</u>




```
<script language="javascript">
var reponse=5<12 && 5<3;
alert("Le résultat du test est : " + reponse);
</script>
```

Le test utilise l'opérateur logique && pour tester les deux conditions simultanément et la réponse de type booléen est ensuite affichée (ici false évidemment).

<u>Exemple 2 : afficher le même test que ci-dessus mais en autorisant le respect de seulement l'une ou l'autre des conditions.</u>

```
<script language="javascript">
var reponse=5<12 || 5<3;
alert("Le résultat du test est : " + reponse);
</script>
```

La même structure que précédemment, seul l'opérateur change et permet alors de ne respecter qu'une seule condition pour renvoyer true. C'est ici le cas (5 est bien inférieur à 12 mais pas inférieur à 3) et c'est pourtant la valeur true qui est renvoyée.

4. Associatifs

Il s'agit de l'opérateur d'affectation égal, qui permet de donner une valeur à une variable (ce qui est à droite du signe égal est affecté à ce qui est à gauche). Cet opérateur peut également être combiné avec les opérateurs de calcul pour réaliser des calculs au moment de l'affectation. La syntaxe est alors la suivante :

```
mavariable+=10 ;
```

Cette instruction permet d'ajouter dix, directement, au contenu de mavariable. Elle remplace astucieusement l'instruction :

```
mavariable=mavariable+10 ;
```

<u>Exemple : modifier le contenu d'une variable égale à 10 en y ajoutant 10 et en utilisant deux techniques : d'abord en passant par l'opérateur arithmétique, puis en utilisant l'opérateur associatif.</u>

```
<script language="javascript">
mavariable=10;
alert("voici le contenu de ma variable : "+mavariable);
mavariable=mavariable+10;
alert("voici le contenu de ma variable : "+mavariable);
mavariable+=10;
alert("voici le contenu de ma variable : "+mavariable);
</script>
```

L'utilisation des deux méthodes simultanément démontre bien le gain obtenu par l'utilisation de l'opérateur associatif.

5. Concaténation

Il existe un seul opérateur de concaténation, c'est le symbole plus « + » qui permet de souder des chaînes de caractères entre elles. C'est grâce à cet opérateur qu'il est possible de construire une chaîne composée de variables et d'un texte, saisi dans le code du script.

Opérateurs	Nom	Rôle	Exemple	Résultat
+	Plus	Concatène les chaînes de caractères.	Bon+jour	Bonjour

L'opérateur de concaténation se distingue, ici, de l'opérateur arithmétique alors que le symbole « + » est le même. Il est donc important de les différencier à l'oral, en utilisant les termes opérateurs de concaténation ou arithmétique.

Exemple : afficher une concaténation de deux variables correspondant au nom et au prénom de l'utilisateur en les séparant d'un espace :




```
<script language="javascript">
var prenom=prompt("Quel est votre prénom ? :", "veuillez saisir
votre prénom ici");
var nom=prompt("Quel est votre nom ? :", "veuillez saisir votre nom ici");
var nomcomplet=prenom+" "+nom;
alert("Votre nom complet est : "+nomcomplet);
</script>
</head>
```

Tout d'abord, il s'agit de demander à l'utilisateur son prénom et son nom par l'intermédiaire de la méthode prompt(). Il faut ensuite effectuer la concaténation des deux chaînes dans une nouvelle variable correspondant au nom complet, tout en veillant à ajouter un espace entre les deux. Pour ce faire, il suffit d'utiliser " " pour le symboliser. Pour terminer, le résultat est affiché dans une boîte de dialogue par la méthode alert().

En plus de ces opérateurs classiques, il existe un certain nombre d'opérateurs spéciaux.

6. Les opérateurs spéciaux

Ce sont des opérateurs n'appartenant à aucune des catégories précédemment citées et dont l'usage est un peu particulier.

Opérateurs	Nom	Rôle	Exemple	Résultat
delete	Delete	Supprime une variable.	delete mavariable	Mavariable n'existe plus
new	New	Crée une instance d'un objet.	new mavariable	Crée une nouvelle instance de mavariable.
this	This	Permet de se référencer à l'objet courant.	this mavariable	Permet de ne pas reciter mavariable.
		Permet de signifier une seule fois un objet pour		Permet de citer une fois ma

with	With	accéder par la suite à plusieurs de ses propriétés ou méthodes.	with mavariable	variable pour en modifier les propriétés.
typeof	TypeOf	Permet de connaître le type d'un objet.	typeof mavariable	Extrait le type de variable de mavariable.
void	void	Exécute une instruction mais ne renvoie pas de valeur.	void(mavariable)	Renvoie undefined.
?:	Opérateur conditionnel ternaire	Permet d'affecter une valeur à une variable selon un test situé à gauche de « ? ». Si le test est vrai, la valeur située à gauche de « : » est renvoyée, sinon c'est celle de droite.	(mavariable<0) ? true :false	Renvoie true si la valeur de mavariable est inférieure à zéro, false dans l'autre cas.

Le cas de l'opérateur conditionnel ternaire sera détaillé au chapitre Contrôler les scripts avec les structures de contrôles.

Exemple : afficher le contenu d'une variable avant et après suppression.

```
<script language="javascript">
mavariable="ma variable n'est pas vide";
alert("La valeur de ma variable est : " + mavariable);
delete mavariable;
alert("La valeur de ma variable est : " + mavariable);
</script>
```

Le script débute par l'affichage de la variable par la méthode alert() pour vérifier son contenu. Puis, l'opérateur spécial delete est utilisé pour supprimer cette variable. La dernière étape consiste à demander l'affichage de la variable qui, comme elle n'existe plus, ne s'effectuera pas et provoquera une erreur.

Le fonctionnement de l'opérateur delete oblige à ne pas utiliser la déclaration explicite des variables par le mot clé var.

Exemple : afficher dans la page le type des variables utilisées dans le script.



```
<script language="javascript">
var numerique=15;
var texte="ma variable";
var booleen=false;
document.write("La variable numerique est de type : "+
typeof numerique +"<BR>");
document.write("La variable texte est de type : "+
typeof texte +"<BR>");
document.write("La variable booleen est de type : "+
typeof booleen +"<BR>");</script>
```

L’affectation de trois variables de type différent, débute le script. Ensuite, l’opérateur typeOf permet d’afficher le type de variable dans une boîte de dialogue.

7. Ordre des opérateurs

Dans le cas où plusieurs opérateurs sont utilisés dans une expression, ils ne sont pas forcément traités dans le sens traditionnel de lecture gauche droite. Le sens de traitement dépend d’un ordre défini dans le core (cœur) de JavaScript et qui correspond au tableau ci-dessous :

Plus le rang est élevé, plus la priorité est importante.

Numéro de rang	Symbole	Type
Rang 15	() [ ].	Appel de fonction / membre
Rang 14	! ~ - ++ --	Négation / complément
Rang 13	* / %	Produit / quotien
Rang 12	+ -	Addition / soustraction
Rang 11	<< >> >>>	Décalage
Rang 10	< <= > >=	Comparaison
Rang 9	== !=	Comparaison
Rang 8	&	ET bit à bit
Rang 7	^	OU exclusif
Rang 6	|	OU inclusif
Rang 5	&&	ET logique
Rang 4	||	OU logique
Rang 3	?:	Conditionnel ternaire
Rang 2	= += -= <<= >>= &= ^= |=	Affectation
Rang 1	,	Séparateur

Exemple : afficher dans une boîte de dialogue le contenu de l’addition de 1 et de 2 multiplié par 3.

```
<script language="javascript">
resultat=1+2*3;
alert("Le résultat du calcul est : "+resultat);
</script>
```

La règle de priorité des opérateurs oblige à faire d’abord la multiplication de 2 par 3, puis d’ajouter 1 au résultat

obtenu.

Exemple : modifier le script précédent pour permettre de faire l'addition puis la multiplication.

```
<script language="javascript">
resultat=(1+2)*3;
alert("Le résultat du calcul est : "+resultat);
</script>
```

Il suffit simplement d'ajouter des parenthèses à l'expression 1+2 pour permettre de changer la priorité des opérateurs.

Incontestablement, les variables sont essentielles dans le déroulement d'un script et celui-ci peut voir son déroulement modifié en fonction de leur valeurs. Ce sont les structures de contrôle qui permettent d'effectuer des tests et des incrémentations de variables.

Les instructions conditionnelles

Ces instructions permettent d'orienter le déroulement du script en fonction de tests. Par exemple, il est possible de contrôler le contenu d'une variable et d'exécuter une instruction si le test est respecté (true), ou d'exécuter une autre instruction, si le résultat du test est faux (false). Ces instructions sont placées à l'intérieur de structures de contrôles, qui permettent de traiter tous les cas de figure en fonction de la réponse du test.

1. if

L'instruction if (si en français) permet de distinguer généralement deux cas de figure seulement. Pour les tests comprenant plus de situations, il est préférable d'utiliser l'instruction switch que nous détaillerons plus loin. Le test doit être placé entre parenthèses et comprend généralement des opérateurs de comparaison et des variables ou constantes. Le test est suivi d'une accolade, la première ligne instruction se trouvant sur la ligne suivante. Il n'y a donc pas de point-virgule à la fin de cette ligne, ce qui correspond à une exception. Il est possible de considérer que la première instruction correspond à un test positif mais il n'y a pas d'obligation. Le nombre de lignes d'instruction est illimité, chacune d'entre elles se terminant par un point-virgule. La gestion du premier cas de figure se termine par une accolade fermante sur la ligne suivante, pour une meilleure lisibilité du code. Le mot clé else (sinon) est ajouté sur la ligne suivante sans point-virgule pour terminer. Une autre accolade est ouverte sur la ligne suivante, elle définit le début des instructions à réaliser si la valeur du test est false. Il est également possible d'ajouter autant de lignes que nécessaires pour traiter ce deuxième cas de figure. Le script continue son déroulement normal après l'accolade fermante. La syntaxe de la structure de contrôle avec l'instruction if est donc la suivante :

```
if (test) {
Ligne 1 d'instruction ;
Ligne 2 d'instruction ;
}
else
{
Ligne 1 d'instruction ;
Ligne 2 d'instruction ;
}
```

Exemple : afficher une boîte de dialogue et tester le réponse pour continuer ou non le déroulement du script.






```
<script language="javascript">
suite=confirm("Voulez-vous poursuivre ? ");
if (suite==true) {
alert("J'en suis heureux ");
}
else {
alert("C'est dommage");
}
</script>
```

Le script débute par l'affichage d'une boîte de dialogue demandant à l'utilisateur s'il souhaite ou pas continuer le déroulement du script. Cette boîte de dialogue récupère la réponse de l'utilisateur dans une variable, appelée suite, par l'intermédiaire de la méthode confirm(). La valeur renvoyée correspond soit à true, soit à false. Si l'utilisateur clique

sur **OK** (true), le script affiche une boîte de dialogue de type alert pour le remercier. Au contraire, si l'utilisateur clique sur le bouton **Annuler** (false dans le test), le script affiche une autre boîte, qui permet de regretter cet abandon.

> À noter les deux signes égal, côte à côte, qui permettent d'effectuer une comparaison et non une affectation.

Il est également possible d'utiliser l'opérateur ternaire pour effectuer le test. Le script devient alors :

```
<script language="javascript">
suite=confirm("Voulez-vous poursuivre ? ");
(suite==true)? alert("J'en suis heureux ") : alert("C'est dommage");
</script>
```

Même si le résultat du script est équivalent au précédent en utilisant cet opérateur, il faut convenir que le décryptage est plus délicat.

Si vous désirez effectuer plusieurs tests sur une variable, il est possible d'imbriquer les " if ".

Exemple : afficher deux boîtes de dialogues successivement et tester leurs réponses pour continuer ou non le déroulement du script, en utilisant des if imbriqués.

```
<script language="javascript">
avis=confirm("Aimez-vous le JavaScript ? ");
suite=confirm("Voulez-vous poursuivre ? ");
if (avis==true) {
      if (suite==true) {
      alert("J'en suis heureux ");
      }
      else {
      alert("C'est dommage");
      }
}
else {
alert("C'est dommage");
}
</script>
```

Nous reprenons le même test que précédemment, mais en ajoutant une question pour savoir si l'utilisateur aime le JavaScript et en affectant la valeur de cette réponse à la variable avis. Si c'est le cas, l'utilisateur reçoit un remerciement uniquement s'il a répondu favorablement aux deux questions. Dans le cas contraire s'il a répondu en cliquant sur annuler pour l'une ou l'autre des questions, le script affiche la seconde boîte de dialogue.

Vous voyez ici l'utilité de l'indentation de script, afin de clarifier la présentation et donc de retrouver facilement les instructions concernant la première et la seconde condition. Mais, vous imaginez facilement la difficulté de lecture d'un script imbriquant cinq ou six if. Cette technique, bien que correcte, est donc à déconseiller fortement. En effet, il est possible de lui préférer l'usage des opérateurs logiques Et « && » et OU « || ».

Exemple : afficher deux boîtes de dialogues successivement et tester leurs réponses pour continuer ou non le déroulement du script, à l'aide des opérateurs &&, || et !=.

```
<script language="javascript">
avis=confirm("Aimez-vous le JavaScript ? ");
suite=confirm("Voulez-vous poursuivre ? ");
if (suite==true && avis==true) {
alert("J'en suis heureux ");
}
else {
alert("C'est dommage");
}
</script>
```

Le même script, en utilisant l'opérateur ou « || » qui permet de ne vérifier qu'une seule des conditions.

```
<script language="javascript">
avis=confirm("Aimez-vous le JavaScript ? ");
suite=confirm("Voulez-vous poursuivre ? ");
if (suite==true || avis==true) {
alert("J'en suis heureux mais cela reste à confirmer");
}
else {
alert("C'est dommage");
}
```

```
</script>
```

Une autre variante utilisant l’opérateur logique Non « != », expliqué au chapitre Utilisation des constantes, variables et opérateurs.

```
<script language="javascript">
avis=confirm("Aimez-vous le JavaScript ? ");
suite=confirm("Voulez-vous poursuivre ? ");
if (suite!=false && avis!=false) {
alert("J’en suis heureux ");
}
else {
alert("C’est dommage");
}
</script>
```

L’utilisation de la structure de contrôle avec if est largement répandue, mais il existe une autre syntaxe à l’aide de l’opérateur conditionnel ternaire.

2. L’opérateur conditionnel ternaire

Pour effectuer des tests, il est également possible d’utiliser l’opérateur conditionnel ternaire. Sa syntaxe est la suivante :

```
(test) ? instruction si vrai : instruction si faux ;
```

Même si la syntaxe est plus concise, la relecture n’est pas facilitée par cette structure de contrôle. C’est pour cette raison que son utilisation est peu fréquente et qu’elle est rarement utilisée dans certains scripts.

Exemple : demander une valeur à l’utilisateur et la tester, pour savoir si elle est supérieure ou inférieure à l’opérateur conditionnel ternaire.






```
<script language="javascript">

var mavar=prompt("Quelle valeur voulez-vous prendre pour faire un test
avec l’opérateur conditionnel ternaire ?");
(mavar<2)? alert("La variable est inférieure à 2"):alert("La variable
est supérieure à 2");

</script>
```

Ici, une valeur est affectée à la variable mavar avec la méthode prompt(), puis le script utilise l’opérateur conditionnel ternaire pour la comparer à 2. Dans le cas où le test est respecté, c’est l’instruction figurant juste après le point d’interrogation qui est exécutée, dans le cas contraire, c’est l’instruction se trouvant juste après le double-point.

3. else if

L'instruction else if est utilisée en complément de if dans le cas où le nombre de situations est supérieur à deux ; nous parlerons alors de concaténation de tests. Le début est identique à la première structure de contrôle, par contre lorsqu'il faut ajouter un second test, il faut utiliser l'instruction else if suivi du nouveau test entre parenthèses et de l'accolade ouvrante. Ainsi, toutes les situations possibles seront traitées par élimination, jusqu'à la dernière. il faut terminer par l'instruction else qui concerne, alors, toutes les autres situations qui n'ont pas été traitées auparavant.

La syntaxe de la structure de contrôle avec l'instruction else if est la suivante :

```
if (test) {
Ligne 1 d'instruction ;
Ligne 2 d'instruction ;
}
else if (test) {
Ligne 1 d'instruction ;
Ligne 2 d'instruction ;}
}
else
{
Ligne 1 d'instruction ;
Ligne 2 d'instruction ;
}
```

Exemple : afficher la valeur d'une réduction obtenue par calcul, en fonction du montant de la commande saisie dans une boîte de dialogue. La réduction sera égale à zéro si le montant de la commande n'atteint pas 10 000 euros. Elle sera égale à 5 % de la commande si celle-ci est comprise entre 10 000 et 25 000 euros et elle sera égale à 10 % au-dessus de ce montant.






```
<script language="javascript">
commande= prompt("Quel est le montant de la vente :", "Saisissez
ici le montant de la commande");
if (commande<10000) {
alert("Le montant de la vente est insuffisant pour prétendre
à une réduction");
}
else if(commande<25000) {
reduction=commande*0.05;
}
else {
reduction=commande*0.1;
}
alert("Le montant de la réduction pour une commande de : " + commande
+ " euros est de : " + reduction + " euros");
</script>
```

Après avoir affecté la variable commande, il est possible d'effectuer des tests sur celle-ci en suivant un ordre logique (ordre croissant ou décroissant). Ici, la logique des tests suivra une logique croissante. Pour débuter, il faut comparer le montant de la commande à 10 000. Si le test est négatif, cela signifie que le montant de la commande n'est pas assez élevé pour obtenir une réduction. L'affichage d'une boîte de dialogue permet de le préciser. Le second cas de ce premier test permet de dire que le montant de la commande est bien supérieur à 10 000 mais est-il supérieur à 25 000 ? Pour le savoir, il faut rajouter un nouveau test après else if. Si la commande est inférieure à 25000, il faut

calculer le montant de la réduction, égale à 5 % de la commande. Dans l'autre cas, le montant de la commande est forcément supérieur à 25 000 et il n'est pas nécessaire d'ajouter un autre test pour le savoir. Le montant de la réduction peut, alors, être calculé et affiché dans une boîte de dialogue.

Lorsqu'ils sont nombreux, les if imbriqués deviennent difficiles à maintenir. Il est très facile de faire une erreur d'alignement des accolades et ainsi de pénaliser la relecture. Dans ce cas, il est toujours possible d'utiliser une autre structure de contrôle. Il s'agit de l'instruction switch.

4. switch

Tout comme la structure comprenant if et else if, l'instruction switch permet de gérer plusieurs cas de figure. Son utilisation peut s'avérer plus facile qu'une concaténation de if avec la série d'accolades ouvrantes et fermantes, dont la lecture est quelquefois délicate. Malheureusement, l'utilisation des opérateurs de comparaison inférieur, supérieur, etc. n'est pas autorisée avec l'instruction switch. Du coup, son utilité peut sembler limitée. Une structure de contrôle avec switch débute avec le mot clé switch suivi de la variable à tester, placée entre parenthèses. La fin de la ligne ne doit pas comporter de point-virgule. À la ligne suivante, il faut ouvrir une parenthèse, puis à chaque valeur possible de la variable, il faut indiquer le mot clé case suivi de la valeur de la variable et de deux points. La ou les ligne(s) suivante (s) correspond(ent) aux instructions à réaliser pour chaque cas. Pour terminer le traitement de chaque cas, il faut utiliser le mot clé break qui permet de quitter la structure de contrôle. La structure de contrôle switch se termine par une accolade fermante.

La syntaxe de la structure de contrôle avec l'instruction switch est la suivante :

```
switch (variable de test)
{
case " premier cas " :
instructions
break ;
case " deuxième cas " :
instructions
break ;
}
```

Exemple : afficher une réponse selon une question posée avec trois propositions de réponse : "Javascript est un langage : A.Non typé B.Faiblement typé C.Typé".




```
<script language="javascript">
reponse=prompt("Javascript est un langage : A.Non typé B.Faiblement
typé C.Typé", "Saisissez ici la lettre correspondant à votre réponse");
switch (reponse)
{
case "A":
alert("Pas tout-à-fait, ré-essayez");
break;
case "B":
alert("Bravo ! C'est exact");
break;
case "C":
alert("Cela n'est pas exact, ré-essayez");
break;
}
</script>
```

Tout d'abord, il faut affecter la variable reponse avec la réponse de l'utilisateur. Ensuite, il faut débuter la structure de contrôle en se basant sur la variable reponse. À chaque cas, il faut indiquer l'affichage d'une réponse différente.

Il faut bien vérifier la présence de break car ce genre d'oubli n'étant pas signalé par le navigateur, vous risquez de perdre beaucoup de temps avant de trouver l'erreur.

Ce script sous-entend que les possibilités de réponse seront respectées (A, B ou C). Mais que se passe-t-il si l'utilisateur choisit de saisir un autre type de réponse ? Le script s'interrompt brutalement sans donner de réponse. Le mieux consiste alors, à utiliser le mot clé default qui correspond à tous les cas qui n'ont pas pu être traités précédemment. Ce mot clé doit être ajouté à la fin du script et suivi de break également. Le script devient alors :

```
<script language="javascript">
reponse=prompt("Javascript est un langage : A.non typé B.Faiblement
typé C.Typé", "Saisissez ici la lettre correspondant à votre réponse");
switch (reponse)
{
case "A":
alert("Pas tout-à-fait, ré-essayez");
break;
case "B":
alert("Bravo ! C'est exact");
break;
case "C":
alert("Pas tout-à-fait, ré-essayez");
break;
default:
alert("Votre réponse ne correspond pas aux propositions");
break;
}
</script>
```

Ainsi dans le cas où l'utilisateur ne saisirait pas une des propositions (A, B ou C), c'est la boîte de dialogue située après default : qui s'affiche.

Dans le cas où un même test doit s'appliquer à plusieurs éléments, il est possible d'intégrer le test dans une structure de répétitions.

Les instructions de répétitions (boucles)

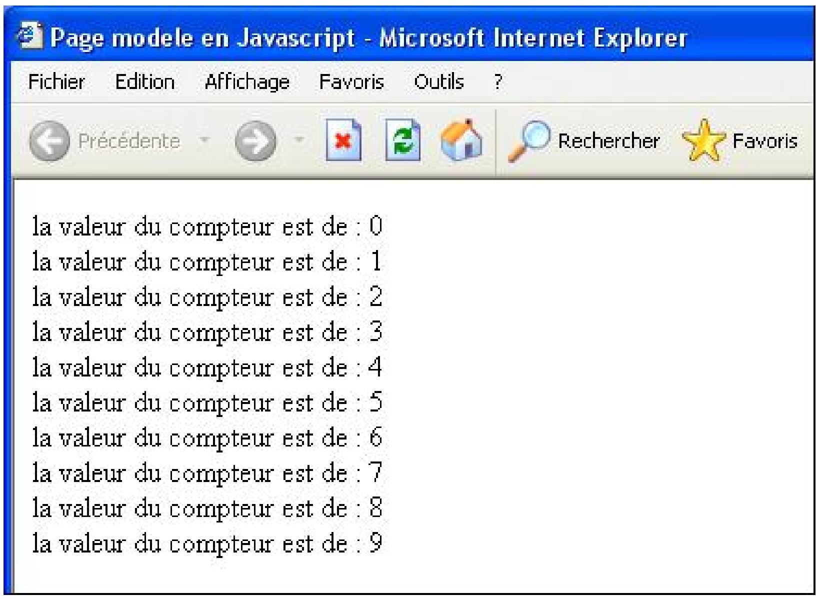
Les boucles permettent de réaliser des blocs d'instructions à plusieurs reprises, appelées instructions d'itération. En Javascript, il en existe deux types, les boucles avec for et les boucles avec while.

1. for

Elle nécessite l'utilisation d'une valeur initiale de compteur, d'un test et d'un facteur de progression. La valeur à atteindre du compteur doit être connue. La syntaxe de cette instruction de répétition est la suivante :

```
for (valeur initiale du compteur ; test de répétition ; facteur de
progression){
instructions à répéter
}
```

Exemple : afficher dix fois dans la page le résultat d'une boucle comportant une variable post-incrémentée.




```
<script language="javascript">
for (compteur=0; compteur<10;compteur++) {
document.write("la valeur du compteur est de : " +
compteur+"<BR>");}
</script>
```

Tout d'abord, il faut paramétrer la structure de boucle en déterminant la valeur de début du compteur, la valeur de fin et l'opérateur d'incrémentation. Il suffit, ensuite, d'insérer dans la boucle, l'affichage de la boîte de dialogue, dans laquelle figure une chaîne de caractères ainsi que le compteur lui-même. Sa visualisation permet de suivre l'incrémentation du compteur à chaque passage. Ici l'affichage se produira 10 fois car le compteur démarre à zéro mais est stoppé à 9 car la condition compteur<10 l'empêche d'aller jusqu'à 10.

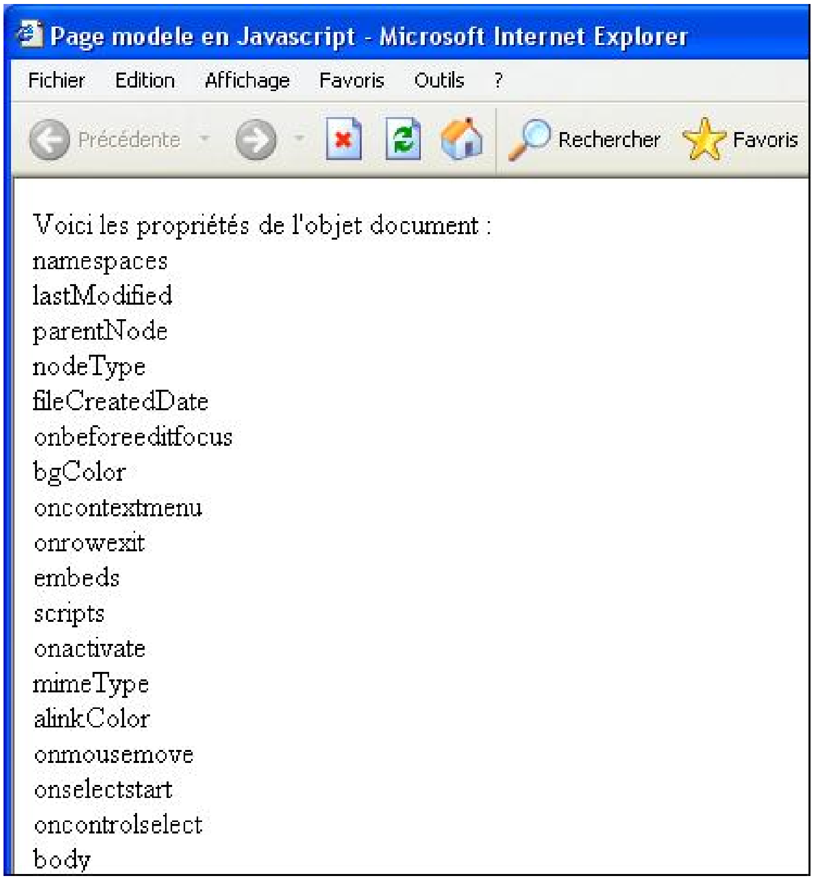
Il est possible d'obtenir le même résultat (10 affichages) en modifiant la valeur d'initialisation de la variable et la valeur limite à atteindre dans le test (par exemple, choisir 1 pour l'initialisation et compteur<11 pour la valeur test).

2. for...in

Ce type de boucle est destiné à manipuler un objet et s'utilise avec la syntaxe suivante :

```
for (var nompropriété in nomdelobjet) {
instructions ;
}
```

Exemple : afficher dans la page la liste des propriétés de l'objet document :




```
<script language="javascript">
document.write("Voici les propriétés de l'objet document : "+"<BR>");
for (var propriete in document) {
document.write(propriete+"<BR>");
}
</script>
```

Certaines propriétés restent masquées par JavaScript et ne pourront par conséquent être listées par ce script. Le retour chariot "\r" ne fonctionne pas avec Firefox.

3. while / do while

Les boucles avec while permettent d'exécuter un bloc d'instructions tant qu'une condition est vraie. Avec while, les instructions figurant entre les accolades ouvrantes et fermantes seront exécutées tant que le test est vérifié. Lorsque le résultat du test prend la valeur false, la boucle s'interrompt et le déroulement du script se poursuit. La syntaxe de la structure de contrôle avec l'instruction while est la suivante :

```
while (test) {
instructions à répéter
}
```

do while permet également la répétition d'instructions à ceci près que le test suit le bloc d'instructions au lieu de le précéder :

```
do {
instructions à répéter
}
while (test);
```

Exemple : afficher une boîte de dialogue demandant la saisie d'un nombre dont la valeur doit être positive. Tant qu'une valeur négative ou nulle est saisie, la boîte de dialogue continue à s'afficher.






```
<script language="javascript">
reponse=-1;
while (reponse<0) {
reponse=prompt("Saisissez une valeur positive", "Saisissez ici
votre valeur");
}
alert(" Merci d’avoir saisi une valeur positive ") ;
</script>
```

La particularité de ce script réside dans le fait que la variable réponse servant de test doit être inférieure à zéro avant de passer sur l’instruction while, ainsi la boîte s’affiche dès le chargement de la page. C’est la raison pour laquelle le script affecte -1 à la variable reponse, dès le début. Lors du premier passage sur l’instruction while, le test est respecté et donc, la boîte de dialogue s’affiche. Par la suite la valeur de reponse devra être supérieure à 0 pour sortir de la structure de contrôle. Si la valeur -1 n’est pas affectée à la variable reponse, celle-ci serait null, donc équivalent à zéro, et ne respecterait pas le test. La boîte de dialogue ne s’afficherait donc pas.

Interrompre et quitter les boucles

1. break

Comme nous l'avons vu précédemment dans la structure de contrôle avec switch, break permet de quitter les boucles de manière prématurée. L'instruction break peut aussi renvoyer le déroulement du script vers une étiquette (un label), c'est-à-dire à une position identifiée par un nom suivi de deux points.

L'instruction break ne peut être utilisée qu'à l'intérieur de boucles imbriquées avec for ou switch.

Le nom de l'étiquette doit suivre les mêmes contraintes de nommage que les variables (pas d'espace, le nom de l'étiquette doit toujours commencer par une lettre, etc.).

L'étiquette doit être placée avant l'instruction break et doit être suivie d'un double point.

Exemple : afficher dans une boîte de dialogue la valeur des compteurs i et j jusqu'à ce que j soit égal à trois.






```
<script language="javascript">
boucle:
for (i=0;i<10;i++) {
alert("La valeur de i est égale à : "+i);
    for (j=0;j<10;j++) {
    alert("La valeur de j est égale à : "+j);
        if (j==3) {
        break boucle;
        }
    }
}
</script>
```

Ici, la première boucle indique la valeur de i, puis la seconde celle de j. Celui-ci apparaît plusieurs fois avant que le test ne soit vérifié. À ce moment, l'instruction break boucle; renvoie le script à l'étiquette boucle, donc en dehors de la boucle correspondant à j et à i. Ensuite, le script reprend son cours après la fin de la plus grande boucle (ici celle qui correspond à i).

Il existe également une instruction qui, contrairement à l'instruction break, permet de poursuivre le déroulement d'un script.

2. continue

L'instruction continue permet de ne pas prendre en compte certaines valeurs qui seraient fausses dans le test, assurant ainsi la poursuite de la boucle.

Exemple : afficher la valeur du compteur d'une boucle de 0 à 4 sauf la valeur 3 :

```
<script language="javascript">
for (j=0;j<5;j++) {
```

```
if (j==3) {
continue;
}
alert("La valeur de j est égale à : "+j);
}
</script>
```

Ici, le script affiche le message d'affichage de la variable j qui s'incrémente à chaque passage de la boucle. Lorsque la variable j est égale à 3, l'instruction continue saute l'instruction d'affichage de la boîte de dialogue. Du coup, la valeur 3 n'est pas affichée, mais le script continue son déroulement et affiche les valeurs restantes.

La gestion des exceptions

JavaScript fournit plusieurs instructions ou structures de contrôles permettant de gérer les erreurs renvoyées par le navigateur lors de l'exécution de script.

1. La structure try...catch

Cette structure permet d'exécuter les instructions placées entre les accolades et correspondants au mot clé try (c'est le bloc d'essai). Dans le cas où une erreur est détectée par le navigateur, les instructions situées entre les accolades correspondant au mot clé catch() sont exécutées. La syntaxe de cette structure est la suivante :

```
try {
Instructions ;
}
catch(identificateur){
Instructions ;
}
```

Exemple : afficher un message indiquant l'impossibilité d'effectuer d'une division pour laquelle aucune variable correspondant au diviseur n'est déclarée.




```
<script language="javascript">
dividende=10;
try {
resultat=dividende/diviseur;
}
catch(exception){
alert("Division Impossible");
}
</script>
```

Dans ce script, le résultat de l'opération dividende/diviseur génère une erreur car il s'agit d'une division par un diviseur indéfini (division par zéro). Le bloc d'instruction try permet de capturer cette erreur et d'orienter le déroulement du script vers le bloc d'instructions catch, permettant l'affichage de la boîte de dialogue.

2. La structure try...finally

Il est possible d'ajouter un bloc final dans le cas où aucune exception n'est détectée. Dans ce cas, il faut inclure les instructions à effectuer dans un bloc délimité par le mot clé finally.

```
try {
Instructions ;
}
catch(identificateur){
Instructions ;
}
finally {
Instructions ;
}
```

Le script devient alors :




```
<script language="javascript">
dividende=10;
try {
resultat=dividende/diviseur;
}
catch(diviseur){
alert("Division Impossible");
}
finally{
diviseur=2;
resultat=dividende/diviseur;
alert("Le résultat de la division est : "+resultat);
}
</script>
```

Dans ce script, l'erreur est bien détectée et l'affichage du message est effectué par le bloc correspondant à catch. Mais le bloc d'instructions finally permet d'affecter la valeur 2 à la variable diviseur, ce qui permet tout de même d'exécuter le script.

Il est possible d'imbriquer les try afin de permettre des tentatives avec des valeurs différentes et ainsi de tester un script sous toutes ses formes.

Rôle des fonctions

Les fonctions font partie des éléments fondamentaux en JavaScript, avec les méthodes dont nous avons déjà parlé. La notion de fonction est assez proche de celle de méthode, car ce sont toutes les deux des procédures qui effectuent une tâche spécifique. La différence réside dans le fait que, les méthodes s'appliquent à un objet particulier alors que les fonctions sont totalement détachées de cette notion. Il existe deux types de fonction, celles prédéfinies, donc déjà intégrées à JavaScript et celles définies par l'utilisateur. Pour les premières, il suffit de les employer au moment voulu sans déclaration préalable. Pour les secondes, il faut impérativement les déclarer avant de pouvoir les appeler plus loin dans le script. Les fonctions prédéfinies sont proches des méthodes attachées aux objets. Le tableau ci-dessous permet d'identifier quelques fonctions utiles :

Fonctions prédéfinies	**Rôle**
`escape()`	Renvoie une chaîne de caractères remplaçant par sa codification ASCII, la chaîne placée entre les parenthèses.
`eval()`	Exécute le code JavaScript placé entre les parenthèses.
`isFinite()`	Teste la valeur placée entre les parenthèses pour savoir si elle correspond ou non à un nombre fini. En JavaScript un nombre est dit fini lorsque sa valeur est supérieure à $1.7976931348623157E+10^{308}$ ou inférieure à $-1.7976931348623157E+10^{308}$.
`isNaN()`	Teste la valeur placée entre les parenthèses pour savoir si elle n'est pas numérique. Renvoie true si le test est vrai, false dans le cas contraire.
`Number()`	Renvoie le résultat d'une conversion en chiffres de la variable placée entre les parenthèses.
`String()`	Renvoie le résultat d'une conversion en texte de la variable placée entre les parenthèses.

Exemple : afficher le résultat d'une conversion en chiffres d'une variable texte :

```
<script language="javascript">
resultat=Number("4")+3 ;
alert("Voici le total avec resultat en texte converti en chiffres :
"+resultat);
</script>
```

Depuis le début de cet ouvrage, les scripts sont insérés dans la page web et exécutés lors de son chargement en mémoire. Même si cela suffit amplement à vérifier pour l'instant leur fonctionnement, cela ne laisse que peu de souplesse dans leur utilisation si bien que les scripts s'exécutent normalement au moment du chargement de la page. Avec les fonctions, il est possible d'exécuter le script à un moment précis, correspondant à un évènement bien déterminé, comme un clic sur un bouton par exemple. Le navigateur lira bien la fonction, mais ne l'exécutera qu'au moment du clic sur le bouton. C'est ce que l'on appelle la programmation évènementielle. Il est aisé de comprendre alors pourquoi le concept de fonction est la base de la programmation objet en JavaScript et qu'il présente ainsi deux avantages :

1. Le regroupement d'instructions dans des blocs nommés, à l'identique des sous-programmes ou des modules.

2. L'exécution de ces blocs d'instructions, à des moments précis, par la programmation évènementielle.

1. La déclaration des fonctions

Pour déclarer une fonction, il faut utiliser le mot clé Function suivi du nom que vous souhaitez lui attribuer puis d'une série de parenthèses, une ouvrante et l'autre fermante, d'une accolade ouvrante { qui indique le début du bloc d'instructions et enfin d'une accolade fermante } délimitant la fin de la fonction.

```
Exemple : Function MaFonction(argument1, argument2) {
Instruction1
Instruction2…
}
```

Attention, les règles de nommage des fonctions suivent les mêmes règles que celles des variables, à savoir :

- Le nom doit commencer par une lettre.

- Le nom peut être composé de lettres, de chiffres et des caractères _ et & mais n'utilisez pas de caractères spéciaux, réservés ou d'espaces.

- Les parenthèses placées à la fin du nom de la fonction permettent de passer des arguments (variables). Même si vous ne passez aucun argument par la fonction, la présence des parenthèses est indispensable. Ne les oubliez, donc, pas.

Les parenthèses permettent de faire passer des arguments. Ces arguments correspondent à un nom et ne sont pas typés. S'il y en a plusieurs, il faut les séparer par une virgule. S'il n'y en a pas, il faut veiller à ajouter les parenthèses tout de même, sous peine de générer une erreur.

Attention, également à ne pas oublier l'accolade fermante.

Rappelez-vous aussi que JavaScript est sensible à la casse et que `MaFonction` n'a rien à voir avec `mafonction`.

Il n'y a pas de limite concernant le nombre de fonctions, par contre il ne peut y avoir plus de 255 arguments (ce qui est largement suffisant). Les accolades permettent de différencier le début et la fin de chaque fonction.

Pour terminer, il vaut mieux que deux fonctions ne portent pas le même nom. Si c'est tout de même le cas, JavaScript prend en compte la dernière occurrence et délaisse les autres fonctions du même nom.

Voici le code d'une fonction JavaScript permettant l'affichage d'un message :

```
<html>
<head>
<title>Fonctions</title>
<script language="javascript">
function MaFonction()
{
alert("Ceci est ma première fonction")
}
</script>
</head>
```

Mais une telle fonction ne peut fonctionner convenablement que si elle est appelée lors d'un évènement particulier. Il faut donc associer à cette fonction un évènement déclencheur.

Une fois la fonction déclarée et créé, il est possible de l'utiliser en l'appelant et en lui fournissant les arguments nécessaires.

Exemple : créer une fonction permettant de calculer le taux d'alcoolémie moyen d'un homme de 80 kg ayant bu deux verres de vin.




```
<script language="javascript">
function tauxalcool(poids,alcool) {
coeff=0.7;
resultat=alcool/poids*coeff;
return resultat;
}
poids=80;
alcool=2*10;
document.write("Le taux d'alcoolémie pour un homme de 80 Kg buvant
deux verres de vin est de : "+tauxalcool(poids,alcool)+"approximativement");
</script>
```

Ici, la fonction calcule le résultat en fonction des arguments qui seront passés, (ici le poids et la quantité d'alcool en grammes). Le résultat sera renvoyé grâce à l'instruction return, qui délimite la fin de la fonction. La suite du script fournit les valeurs servant d'éléments au calcul et appelle la fonction en lui passant la valeur des arguments. La

méthode document.write() permet ensuite, d'afficher dans la page le résultat renvoyé par la fonction.

2. Utiliser plusieurs fonctions et transférer des données entre elles

Une des particularités des fonctions concerne l'usage des variables. En effet, si une variable est déclarée en dehors d'une fonction, elle est disponible pour l'ensemble du script. Elles sont alors appelées des variables globales. Par contre, lorsque les variables sont déclarées à l'intérieur d'une fonction (elles sont alors désignées sous le terme de variables locales), elles ne sont pas accessibles par les autres fonctions.

Ainsi, dans le code suivant la variable `variable1` n'est pas disponible et l'affichage ne pourra donc pas fonctionner.

```
<html>
<head>
<title>Les fonctions</title>
<script language="javascript">
function premierefonction() {
var variable1="Voici ma première variable";
}
function afficher() {
alert(variable1);
}</script>
</head>
<body>
<input type="button" name="Submit" value="Afficher"
onClick="afficher()">
</body>
</html>
```

Pourtant, il est possible de transférer des valeurs entre deux fonctions de manière relativement simple.

Il suffit d'appeler la seconde fonction à partir de la première, en identifiant la variable qui transite entre les deux.

Exemple : transférer des variables de réponse d'une fonction à l'autre :

Nous souhaiterions, par exemple, créer une première fonction destinée à afficher une boîte de dialogue et à recueillir la réponse de l'utilisateur suite à un clic sur l'un ou l'autre des boutons de la boîte.



Cette réponse sera ensuite transférée à une autre fonction, qui aura pour rôle d'afficher une autre boîte de dialogue précisant que la seconde fonction est activée et affichant le résultat issu de la première fonction.






```
<script language="javascript">
function pose_question()
{
var reponse =confirm("Voulez-vous continuer ?");
test_reponse(reponse)
}
function test_reponse(reponse)
{
alert("Nous sommes dans la fonction test_reponse");
alert("La réponse saisie dans la fonction pose_question est : "+
reponse);
}
</script>
```

Ainsi, l'expression `test_reponse(reponse)` permet le passage de données d'une fonction à l'autre par l'appel de la seconde fonction (test_reponse) celle-ci en récupérant la valeur de la réponse donnée en argument.

Ensuite, l'expression de déclaration `function test_reponse(reponse)` permet à la fonction test_reponse de recevoir comme argument la variable reponse qui lui servira pour l'affichage des boîtes de dialogue.

3. L'instruction return

Cette instruction permet de retourner le résultat d'une fonction. Si la fonction n'a rien à renvoyer, la valeur undefined

sera retournée.

Exemple : afficher, dans une boîte de dialogue, le résultat d’une addition, dont le calcul est réalisé dans une fonction nommée total.




```
<script language="javascript">
function total(chiffre1,chiffre2) {
addition=chiffre1+chiffre2;
return addition;
}
var chiffre1=10;
var chiffre2=20;
alert("Le résultat de l’addition de la fonction total est :
"+total(chiffre1,chiffre2));
</script>
```

Une fonction peut également être affectée à une variable. Il suffit pour cela de déclarer la variable et de lui affecter la fonction par le signe égal.

Exemple : créer une variable nommée total correspondant à une fonction contenant deux paramètres (chiffre1 et chiffre2 respectivement égal à 10 et 20).




```
<script language="javascript">
var total = function(chiffre1,chiffre2) {
addition=chiffre1+chiffre2;
return addition;
}
alert(" Le résultat de l’addition de la fonction total est : "+ total(10,20));
</script>
```

4. Les closures

Une des particularités de JavaScript réside dans le fait que ce langage supporte les closures. Cet anglicisme (traduisible en français par fermeture) signifie qu’une fonction A peut comporter une autre fonction B qui, elle-même, fait référence à la fonction A.

Exemple : afficher le résultat d’une addition prenant la valeur de deux variables, issues de fonctions différentes.




```
<script language="javascript">
```

```
function externe(parametre) {
var variable1 = parametre;
function interne() {
var variable2=10;
resultat=variable1+variable2;
alert("L'addition des deux variables provenant de deux fonctions
différentes est : "+resultat);

}
return interne;
}
var affiche= externe(1);
affiche();
</script>
```

Le script se divise en deux fonctions différentes. La première fonction appelée externe dispose d'un paramètre (dans cet exemple, égal à 1) transmis par l'instruction var affiche= externe(1);.

Ce paramètre est, ensuite, transmis à la variable nommée variable1. La fonction interne qui suit a pour objectif de renvoyer et d'afficher le résultat de la variable1 issue de la fonction parente nommée externe et de la variable2 définie en son sein. L'ensemble du traitement est enfin effectué, lors de l'appel de la variable affiche, qui n'est elle-même qu'un appel à la fonction externe().

Attention cependant, les closures peuvent provoquer des fuites de mémoire néfastes pour le déroulement des scripts en ralentissant, voire en bloquant le traitement du navigateur.

5. Fuites de mémoire

Une fuite de mémoire correspond à une sur-utilisation des capacités de la mémoire de l'ordinateur. En règle générale, elle est involontaire et résulte du fait que le navigateur ne libère pas la mémoire dont il n'a plus besoin. Ce problème est d'autant plus crucial avec le web 2.0 et l'utilisation d'Ajax. En effet, en utilisation classique (c'est-à-dire sans Ajax), les pages se succèdent et le garbage collector de JavaScript (ramasse-miettes en français) effectue un travail de libération de la mémoire des objets, qui ne seront plus utiles par la suite. Avec Ajax, les données composant la page, peuvent se multiplier puisqu'il n'y a plus forcément de changement de page. Internet Explorer est nettement plus concerné par ce problème, du fait qu'il utilise son propre système de garbage collector, qui peut entrer en conflit avec celui de JavaScript. En définitive, les closures constituent une des principales causes de génération de fuites de mémoire.

6. Le mot clé This

Ce mot clé permet d'identifier l'objet sur lequel vous désirez travailler (on parle d'instance de l'objet).

Utiliser une fonction pour créer un objet

En JavaScript comme dans la plupart des langages orientés objet, il est possible de ne pas se limiter aux objets natifs de JavaScript et de créer ses propres objets.

Il existe deux façons de construire un objet en JavaScript. Il est possible de le faire directement en initialisant un objet ou en passant par l'intermédiaire d'un prototype.

La méthode du prototypage est la plus courante et la plus simple à utiliser. Pour y parvenir, il faut suivre deux étapes successives.

Tout d'abord, il faut créer une fonction qui sert de constructeur au nouvel objet et de déclaration des différentes propriétés. Cela s'appelle ici, un prototypage car il s'agit bien de construire un prototype d'objet.

```
Function nomdemonobjet(propriété1, propriété2, propriété3){
This.propriété1=valeurPropriété1 ;
This.propriété1=valeurPropriété2 ;
This.propriété1=valeurPropriété3 ;
```

Vous pouvez ensuite réutiliser l'objet à l'aide des mots clés var et new :

```
Var monobjet=new nomdemobjet("valeur1","valeur2","valeur3") ;
```

<u>Exemple : afficher, dans une boîte de dialogue, les informations d'un objet personnel appelé voiture et caractérisé par une marque, un modèle et une année.</u>




```
<script language="javascript">
function voiture(propMarque, propModele, propAnnee) {
this.marque=propMarque;
this.modele=propModele;
this.annee=propAnnee;
}
var mavoiture=new voiture("Peugeot","307","2007");
alert("Voici les caractéristiques de mon objet voiture :" + "\r
la marque : " + mavoiture.marque + "\r le modèle : " + mavoiture.modele
+ "\r l'année : " + mavoiture.annee);
</script>
```

Ici, l'objet voiture dispose de trois propriétés, (propMarque, propModele et propAnnee), correspondant à "Peugeot" ; "307" ; "2007". Il est alors possible de créer une instance de l'objet avec ces valeurs. Ensuite, il ne reste plus qu'à afficher le message dans une boîte de dialogue en ajoutant les propriétés de l'objet, à l'endroit désiré.

Pour rappel, l'utilisation de \r ne fonctionne pas avec Firefox/Mozilla, par conséquent, les informations apparaissent sur une seule ligne.

Les fonctions et la programmation évènementielle sont largement employées, notamment avec les formulaires qui correspondent à des pages contenant des champs à compléter par l'utilisateur.

Les évènements

Les évènements s'appliquent aux objets présents dans la page : les boutons, les champs ou les ascenseurs de la fenêtre mais aussi la page en elle-même. C'est grâce aux évènements que la page devient interactive. Il est possible d'activer une action lorsqu'un évènement se produit. La syntaxe à suivre correspond à :

```
évènement= "actionaréaliser" ;
```

Il suffit, donc, de nommer l'évènement (voir la liste ci-dessous), de le faire suivre du signe égal « = » et de rédiger l'action qui doit lui correspondre.

Objets concernés	Évènement	Se Produit...
window	`OnLoad`	Au chargement de l'objet.
window	`OnUnLoad`	Au déchargement de l'objet.
image, window	`Onabort`	À l'arrêt du chargement de l'objet
window	`OnMove`	Au déplacement de la fenêtre.
window	`OnResize`	Au redimensionnement de la fenêtre.
window	`OnScroll`	Au défilement de la fenêtre par l'ascenseur.
button, checkbox, document, link, radio, reset, select, submit	`OnClick`	Au clic souris.
document, link	`OnDbClick`	Au double clic souris.
button, document, link	`OnMouseDown`	Au maintien de la pression sur le bouton gauche de la souris.
button, document, link	`OnMouseUp`	Au relâchement du bouton de la souris.
area, layer, link	`OnMouseOver`	Au passage de la souris sur un des objets.
aucun	`OnMouseMove`	Au moment du déplacement de la souris.
layer, link	`OnMouseOut`	À la sortie de la souris sur un des objets.
button, checkbox, fileUpload, layer, password, radio, reset, select, submit, text, textArea, window	`OnFocus`	À la réception du curseur dans un des objets.
form	`OnReset`	À la réinitialisation d'un formulaire.
image	`abort`	À l'arrêt du chargement de l'image.
document, image, link, textArea	`Keydown`	Au moment du maintien enfoncé d'une touche du clavier.
document, image, link, textArea	`Keypress`	Au moment de la pression sur une touche du clavier.

document, image, link, textArea	Keyup	Au moment du relâchement d'une touche du clavier.
window	Dragdrop	Au moment d'un glissé déposé dans la fenêtre.
fileUpload, select, submit, text, textArea	OnChange	Au moment d'un changement d'un des objets.
button, checkbox, fileUpload, layer, password, radio, reset, select, submit	Onblur	Au moment de la perte du focus.
image, window	Onerror	Lorsqu'une erreur sur l'objet survient.

Exemple : afficher une boîte de dialogue, au moment du chargement de la page :

```
<script language="javascript">
window.document.OnLoad=alert("Cette fenêtre s'affiche au moment
du chargement de la page");
</script>
```

Mais généralement, le déclenchement de l'action est conditionné par un événement correspondant à un élément HTML de la page comme un bouton ou un champ de formulaire. Afin d'affecter une action à cet élément HTML, il faut :

- inclure les actions dans une fonction comme nous l'avons vu précédemment ;
- affecter un évènement (onClick, onChange...) à l'élément HTML (élément présent dans la partie BODY) ;
- activer la fonction (présente, elle, dans la partie HEAD), en la plaçant entre guillemets.

La syntaxe correspond alors à :

```
<ObjetHTML évènement= "mafonction()" ;/>
```

Si la fonction n'attend pas d'arguments, il faut, tout de même, laisser les parenthèses ouvrante et fermante. Au contraire, si celle-ci attend des arguments, il faut les faire figurer entre ces parenthèses.

Évidemment pour que cela fonctionne correctement, il faut que la fonction soit rédigée avant son appel.

Exemple : en reprenant l'exemple précédent, associer l'affichage d'une boîte de dialogue au clic sur un bouton.




```
<html>
<head>
<title>Les Fonctions et les évènements</title>
<script language="javascript">
function MaFonction()
{
alert("Ceci est ma première fonction")
}
</script>
</head>
<body>
<form id="form1" name="form1" method="post" action="" >
```

```
 <p>
    <input type="submit" name="Submit" value="Fonctionner"
onClick="MaFonction()" />
  </p>
</form>
</body>
</html>
```

Tant que le visiteur ne clique pas sur le bouton, la fonction ne s’exécutera pas.

Il est possible, également, d’utiliser les évènements pour passer des arguments aux fonctions.

Exemple : passer la valeur affectée à un bouton comme argument à une fonction, permettant d’afficher son nom.




```
<html>
<head>
<title>Affiche le nom du bouton</title>
<script language="javascript">
function affiche(nombouton) {
alert("Vous avez cliqué sur le bouton : "+nombouton);
}
</script>
</head>
<body>
<input type="button" name="Bouton" value="Nom"
onclick="affiche(this.value)" />
</body>
</html>
```

Ici, le bouton nommé **Bouton** dispose d’une valeur égale à la chaîne de caractères « Nom », qui est renvoyée à la fonction affiche comportant un argument nommé **nombouton**. La fonction affiche() peut, alors, afficher dans une boîte de dialogue la valeur renvoyée par le bouton.

L’utilisation des évènements est, donc, directement liée aux fonctions.

Utilisation de JavaScript avec les formulaires

L'utilisation de JavaScript pour les formulaires est l'utilisation la plus fréquente. En effet, il est indispensable de contrôler la validité des champs du formulaire pour détecter des saisies non-conformes, qui seraient à l'origine d'erreurs, lors de transmission des informations à une base de données.

Les cas les plus généralement rencontrés sont, par exemple, la saisie d'une adresse e-mail non-conforme ou l'absence de saisie dans un champ obligatoire.

En plus du contrôle des saisies effectuées, il est également possible de procéder à des calculs entre plusieurs champs numériques d'un formulaire pour déterminer, par exemple, un sous-total à partir d'une quantité et d'un prix unitaire.

Le dernier exemple présenté dans ce chapitre nous permettra de parcourir l'ensemble de ces fonctionnalités par la réalisation d'un formulaire de réservation de plusieurs produits, sa transmission et son impression.

Pour débuter, examinons l'objet Form.

L’objet Form

Toute création de formulaire nécessite l’ajout, dans la page HTML, d’un objet dénommé « Form » contenant les champs de texte, les listes déroulantes et autres contrôles comme les boutons radio ou les cases à cocher.

L’objet Form est un sous-objet de l’objet document dans la hiérarchie des objets JavaScript. Il dispose de propriétés et de méthodes qui permettent de le manipuler ou de lui affecter des actions spécifiques (effacement du contenu de tous les champs du formulaire, envoi par e-mail des informations saisies…).

Détaillons quelques propriétés et méthodes utiles à la manipulation de l’objet Form.

1. Propriétés

`action`

Correspond à l’action qui devra être exécutée par le formulaire (généralement il s’agit de mailto :).

`encoding`

Définit le type de codage des données à employer lors de l’envoi du formulaire (par exemple, text/plain permet d’indiquer que les données seront envoyées sous la forme de texte).

`length`

Correspond au nombre de formulaires dans la page.

`method`

Correspond à la méthode d’envoi du formulaire (Get ou Post).

`name`

Correspond au nom du formulaire.

`target`

Désigne la fenêtre cible d’un jeu de cadres devenant active après l’envoi d’un formulaire.

2. Méthodes

`handlEvent()`

Permet de centraliser le traitement vers un seul gestionnaire (méthode non reconnue par Internet Explorer).

`Reset()`

Efface le contenu des champs du formulaire.

`Submit()`

Envoie le formulaire.

Les éléments de formulaire

Les éléments de formulaire représentent les champs, les cases à cocher, les boutons radio ou les listes déroulantes, permettant de récupérer les informations saisies ou choisies.

Le placement des éléments de formulaire est rendu plus facile en utilisant un tableau sans quadrillage, cela permet d'aligner les étiquettes de description dans une première colonne et les champs de formulaire dans une seconde. Le tableau est lui-même intégré dans le formulaire.

Pour effectuer des contrôles, il est nécessaire de rédiger une fonction JavaScript à placer dans la partie HEAD de la page. Cette fonction est appelée au moment du clic sur le bouton du formulaire, par l'instruction onClick= "controle()".

Pour mémoire, voici un tableau permettant de faire la correspondance entre les balises HTML et les objets JavaScript :

Balise HTML	Objet JavaScript	Description
<INPUT TYPE ="text">	text	Zone de saisie monoligne
<TEXTAREA>	textarea	Zone de saisie multilignes
< INPUT TYPE ="checkbox">	checkbox	Case à cocher
< INPUT TYPE ="radio">	radio	Bouton radio
<SELECT>	select	Zone de sélection
< INPUT TYPE ="submit">	submit	Permet l'envoi des données
< INPUT TYPE ="reset">	reset	Réinitialise le formulaire
< INPUT TYPE ="password">	password	Zone de mot de passe
< INPUT TYPE ="hidden">	hidden	Champ caché
< INPUT TYPE ="file">	fileUpload	Zone de sélection de fichier

JavaScript permet d'effectuer des contrôles sur ces objets :

1. Manipulation de champ text

Les champs text permettent de recevoir des valeurs saisies par l'utilisateur, mais également d'afficher des résultats (par exemple, ceux obtenus à la suite de calculs).

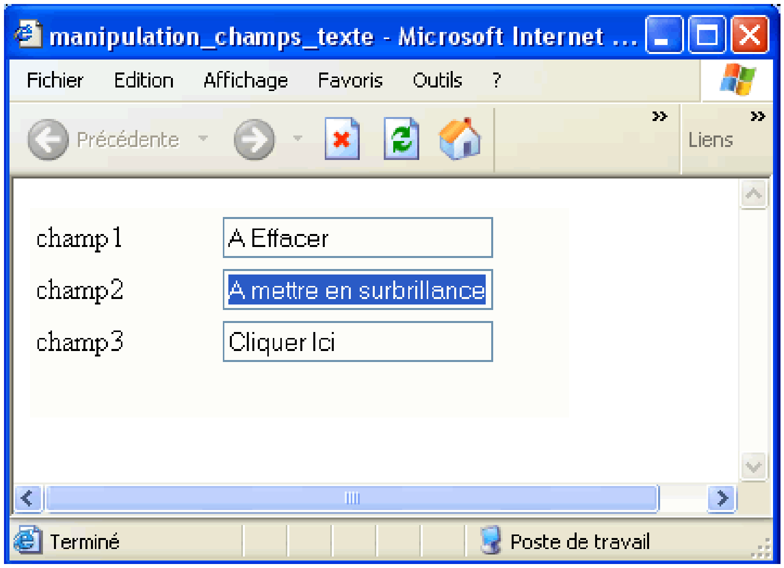
Le champ text est monoligne, il ne suffit pas à une importante chaîne de caractères. Dans ce dernier cas, il est souhaitable d'utiliser textarea.

Pour accéder à la valeur d'un champ text de formulaire, il faut utiliser la syntaxe pointée. Ainsi, pour accéder à la valeur saisie dans le champ1 du formulaire Form1 et la stocker dans la variable ma_variable, il faudra utiliser la syntaxe suivante : `ma_variable=document.Form1.champ1.value` où document est facultatif.

Propriétés	Description
value	Correspond aux informations saisies dans le champ.
defaultValue	Correspond à la valeur s'affichant par défaut dans le champ texte.
Méthodes	**Description**
select()	Affiche en vidéo inverse le champ texte où se trouve le curseur.

`blur()`	Libère le champ texte du curseur.
`focus()`	Positionne le curseur sur le champ texte.

Exemple 1 : effacer le contenu d'un champ texte et en sélectionner un autre, en cliquant dans un troisième champ.

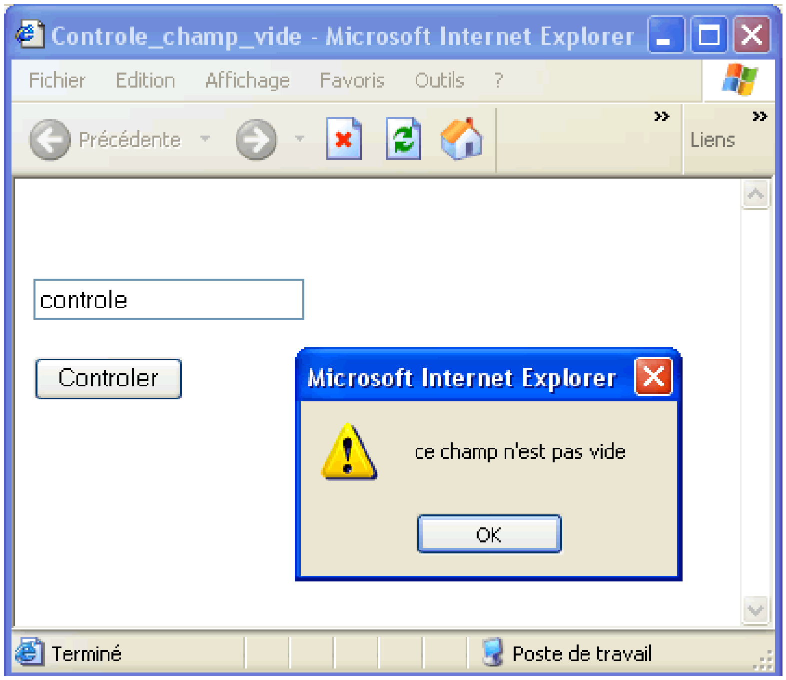



```
<html>
<head>
<title>manipulation_champs_texte</title>
<script language="javascript">
function manipchamptexte()
{
form1.champ1.value='';
form1.champ2.select();
}
</script>
</head>
<body>
<form id="form1" name="form1" method="" action="" >
  <table width="400" border="0">
    <tr>
      <td>champ1</td>
      <td><input name="champ1" type="text" id="champ1" value=
"A Effacer"></td>
    </tr>
    <tr>
      <td>champ2</td>
      <td><input type="text" id="champ2" value="A mettre en
surbrillance"></td>
    </tr>
    <tr>
      <td>champ3</td>
      <td><input name="champ3" type="text" id="champ3"
onFocus="manipchamptexte()" value="Cliquer Ici"></td>
    </tr>
  </table>
  </form>
</body>
</html>
```

Dans un formulaire comportant trois champs textes, en cliquant sur le champ 3, vous effacez le contenu du champ 1 et vous mettez en surbrillance le contenu du champ 2.

L'évènement déclencheur de la fonction manip_champ_texte() est ici le clic sur le champ 3, identifié par onFocus.

Exemple 2 : contrôler si un champ Text est vide par une fonction lancée après un clic sur bouton, puis afficher un message d'avertissement dans une boîte de dialogue :




```
<html>
<head>
<title>Controle champ text</title>
<script language="javascript">
function controlevide()
{
if(form1.nom.value ==’’) {
alert("ce champ est vide");
}
else {
alert("ce champ n’est pas vide");
}
}
</script>
</head>
<body>
<form id="form1" name="form1" method="" action="" >
  <p>   </p>
  <p>
    <input name="nom" type="text" id="nom">
  </p>
  <p>
    <input type="submit" name="Submit" value="Envoyer"
onClick="controlevide()" />
      </p>
</form>
</body>
</html>
```

La fonction controlevide débute par un test de la valeur `if(form1.nom.value ==’’)` d’un champ de type text pour savoir s’il est vide. Si c’est le cas, une boîte de dialogue rappelant que la saisie est obligatoire s’affiche, sinon une autre boîte de dialogue s’affiche confirmant la saisie.

L’instruction `if(form1.nom.value ==’’)` permet de tester la valeur du champ de texte et de savoir si il est vide. Dans ce cas, la boîte de dialogue indiquant cet état s’affiche, sinon une autre boîte confirme que le champ n’est pas vide.

2. Manipulation de champ textarea

Les champs de type textarea sont identiques aux champs de type text, mais ils sont utilisés dans le cas où la chaîne de caractères saisie est trop longue pour un champ text.

Propriétés	**Description**

value	Valeur de l'élément.
defaultValue	Correspond à la valeur s'affichant par défaut dans le champ textarea.

Exemple : concaténer le contenu de trois champs text dans un champ textarea.




```
<html>
<head>
<title>manipulation_champs_textearea</title>
<script language="javascript">
function manipchamptextearea()
{
form1.Total_champ.value=form1.champ1.value+form1.champ2.value
+'\n'+form1.champ3.value;
}
</script>
</head>
<body>
<form id="form1" name="form1" method="" action="" >
  <table width="400" border="0">
    <tr>
      <td>champ1</td>
      <td><input name="champ1" type="text" id="champ1"></td>
    </tr>
    <tr>
      <td>champ2</td>
      <td><input type="text" id="champ2"></td>
    </tr>
    <tr>
      <td>champ3</td>
      <td><input name="champ3" type="text" id="champ3"></td>
    </tr>
    <tr>
      <td>Total champ </td>
      <td><textarea name="Total_champ" cols="15" rows="2"
id="Total_champ"></textarea></td>
    </tr>
    <tr>
      <td colspan="2"><input type="button" name=" Concatener "
value="Concatener" onClick="manipchamptextearea()"></td>
    </tr>
  </table>
</form>
</body>
```

```
</html>
```

La fonction manip_champ_textarea() fait la concaténation des deux premiers champs du formulaire, puis écrit la valeur du troisième champ, en faisant un retour à la ligne entre ces deux parties grâce à Newline (\n).

Ici, il s'agit simplement d'utiliser l'opérateur de concaténation + pour obtenir une nouvelle chaîne de caractères. La fonction est exécutée au moment du clic sur le bouton **Concatener**.

3. Contrôle des cases à cocher

Pour les cases à cocher, il est possible de savoir si elles sont cochées ou non et, donc d'orienter le déroulement du script.

Il est possible de faire plusieurs réponses en cochant plusieurs cases, pour une même question.

Propriétés	Description
checked	Valeur booléenne égale à true lorsque la case est cochée et false lorsqu'elle ne l'est pas.
defaultChecked	Valeur booléenne correspondant à la case cochée (c'est-à-dire avec la valeur true) au moment du chargement du formulaire (c'est la valeur par défaut).

Exemple : tester les réponses, sous forme de cases à cocher possibles, à une question et afficher une boîte de dialogue.

Dans cet exemple il y a quatre possibilités de réponse :

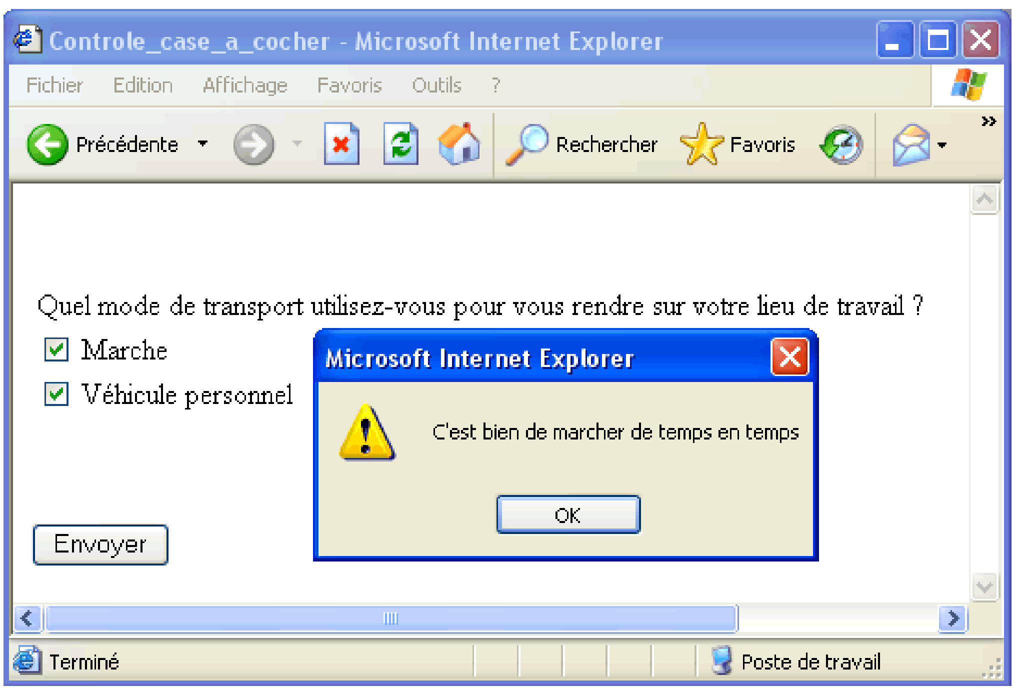
Aucune des cases n'est cochée.

La première case est cochée.

La seconde case est cochée.

Les deux cases sont cochées.

Pour ce dernier cas, il faudra associer les deux valeurs simultanément dans le test par le « et logique » (&&).




```
<html>
<head>
<title>Controle_case_a_cocher</title>
<script language="javascript">
function controlecaseacocher()
{
```

```
if((form1.marche.checked ==true) && (form1.vp.checked ==true))
alert("C'est bien de marcher de temps en temps");
else if(form1.vp.checked ==true)
alert("Ce n'est pas bon pour l'environnement");
else if(form1.marche.checked ==true)
alert("La marche est bonne pour la santé");
else alert("Répondez en cochant au moins une case");
}
</script>
</head>
<body>
<form id="form1" name="form1" method="" action="" >
  <p>   </p>
  <table width="643" border="0">
    <tr>
      <td width="633">Quel mode de transport utilisez-vous pour vous
rendre sur votre lieu de travail ?      </td>
    </tr>
    <tr>
      <td><input name="marche" type="checkbox" id="marche" value="checkbox">
        Marche </td>
    </tr>
    <tr>
      <td><input name="vp" type="checkbox" id="vp" value="checkbox">
      V&eacute;hicule personnel </td>
    </tr>
  </table>
  <p> </p>
  <p>
    <input type="submit" name="Submit" value="Envoyer"
onClick="controlecaseacocher()" />
      </p>
</form>
</body>
</html>
```

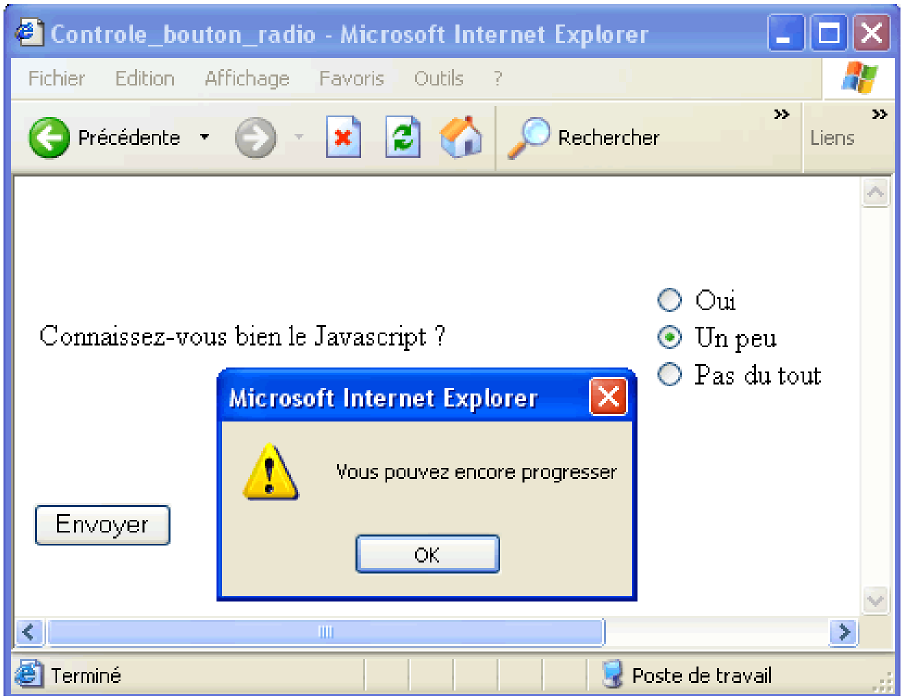
Il est nécessaire d'appliquer quatre tests différents pour traiter les quatre situations possibles. La validité du test permettra d'afficher la boîte de dialogue correspondante.

4. Contrôle des boutons radio

Les boutons radio doivent porter le même nom afin de pouvoir les identifier en fonction de leur numéro d'index. Ainsi, le premier bouton radio est identifié par 0, le second par 1 et le troisième par 2.

Lorsqu'un clic sélectionne un bouton, tous les autres sont désélectionnés ; ce qui différencie l'utilisation des boutons radio de celle des cases à cocher.

Propriétés essentielles	Description
`checked`	Valeur booléenne égale à true lorsque le bouton est activé et false lorsqu'il ne l'est pas.
`defaultChecked`	Valeur booléenne correspondant au bouton activé (c'est-à-dire avec la valeur true) au moment du chargement du formulaire (c'est la valeur par défaut).



Exemple : tester les réponses à une question sous forme de trois boutons radio.

```
<html>
<head>
<title>Controle_bouton_radio</title>
<script language="javascript">
function controleboutonradio()
{
if(form1.connaitre_Javascript[0].checked) {
alert("C'est très bien");
}
if (form1.connaitre_Javascript[1].checked) {
alert("Vous pouvez encore progresser");
}
if (form1.connaitre_Javascript[2].checked) {
alert("Nous n'en sommes qu'au début");
}
}
</script>
</head>
<body>
<form id="form1" name="form1" method="" action="" >
  <p>   </p>
  <table width="617" border="0">
    <tr>
      <td width="326">Connaissez-vous bien le Javascript ? </td>
      <td width="281"><p>
        <label>
          <input type="radio" name="connaitre_Javascript" value="Oui">
          Oui</label>
        <br>
        <label>
          <input type="radio" name="connaitre_Javascript" value="Un peu">
          Un peu</label>
        <br>
        <label>
          <input type="radio" name="connaitre_Javascript" value=
"Pas du tout">
          Pas du tout</label>
      </p></td>
    </tr>
  </table>
  <p> </p>
  <p>
```

```
    <input type="submit" name="Submit" value="Envoyer"
onClick="controleboutonradio()" />
  </p>
</form>
</body>
</html>
```

La fonction controle_bouton_radio teste les boutons radio, appelés connaitre_Javascript suivis du numéro d'index entre crochets. En fonction de la propriété checked, un message s'affiche dans une boîte de dialogue.

5. Contrôle des valeurs d'une sélection de liste

Le contrôle des valeurs d'une liste se rapproche de celui des valeurs des cases à cocher, les éléments de la liste étant identifiés également par un numéro d'index. L'élément choisi est simplement identifié par l'instruction selectedIndex.

Propriétés essentielles	**Description**
`length`	Correspond au nombre de valeurs dans la liste.
`selectedIndex`	Correspond à une valeur dans la liste, identifiée par son numéro dans l'index. L'index commence à zéro.
`defaultSelected`	Valeur par défaut sélectionnée dans la liste.

<u>Exemple : ouvrir des boîtes de dialogue permettant d'afficher un message différent, en fonction d'une sélection faite dans une liste déroulante.</u>




```
<html>
<head>
<title>Controle_liste</title>
<script language="javascript">
function controleliste()
{
if(form1.Question.selectedIndex ==0)
alert("C'est moins");
else if(form1.Question.selectedIndex==1)
alert("C'est plus");
else {
alert("C'est la bonne vitesse");
}
}
</script>
</head>
<body>
<form id="form1" name="form1" method="" action="" >
```

```
  <p>Quelle était la vitesse maximum du TGV lors de son record
le 3 avril 2007 ? </p>
  <select name="Question" id="Question">
    <option>630,1 Km/h</option>
    <option>477,8 Km/h</option>
    <option>574,8 Km/h</option>
  </select>
  <p> </p>
  <p>
    <input type="submit" name="contrôler" value="contrôler"
onClick="controleliste()" />
  </p>
</form>
</body>
</html>
```

Le script s'exécute au moment du clic sur le bouton **contrôler**. La fonction controleliste teste l'indice de la sélection dans la liste (le premier indice étant identifié par le numéro zéro). En fonction de cette sélection, le script affiche une boîte de dialogue.

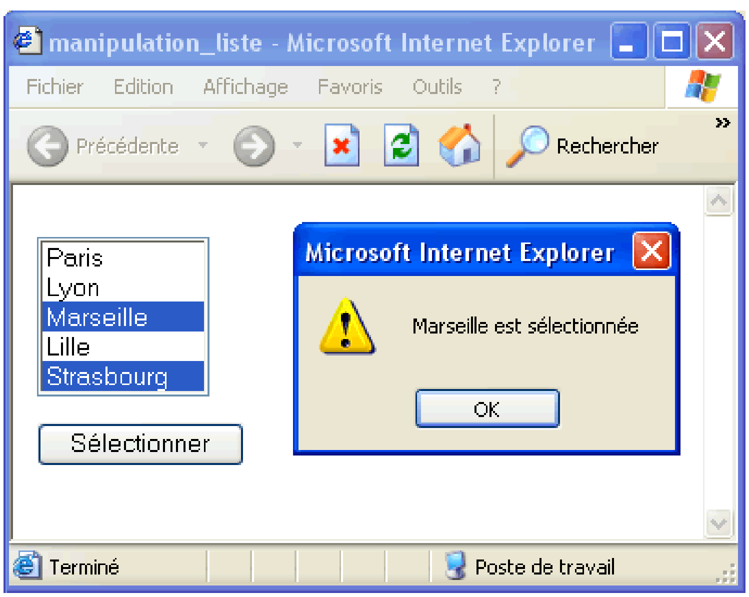
6. Contrôle des valeurs d'une liste à sélections multiples

Les listes à sélections multiples permettent de sélectionner plusieurs valeurs de la liste en utilisant la touche [Alt] pour les sélections contiguës et la touche [Ctrl] pour les sélections qui ne le sont pas.

Le script va contenir une boucle sur les éléments de la liste, qui contrôlera pour chacun si leur valeur a été sélectionnée.

Propriétés essentielles	Description
`length`	Correspond au nombre de valeurs dans la liste.
`selectedIndex`	Correspond à une valeur dans la liste identifiée par son numéro dans l'index. L'index commence à zéro.
`defaultSelected`	Valeur par défaut sélectionnée dans la liste.

Exemple : afficher dans des boîtes de dialogue successives, les valeurs sélectionnées dans une liste à sélection multiple.




```
<html>
<head>
<title>manipulation_liste_multiple</title>
<script language="javascript">
function maniplistemultiple()
```

```
{
var ville="";
nb=form1.liste_ville.length;
i=form1.liste_ville.selectedIndex;
for (i = 0;i<nb;i++){
if(form1.liste_ville.options[i].selected){
ville=form1.liste_ville.options[i].value;
alert(ville+" est sélectionnée");
}
}
}
</script>
</head>
<body>
<form id="form1" name="form1" method="" action="" >
  <table width="146" border="0">
    <tr>
      <td width="140" height="108"><select name="liste_ville"
size="5" multiple id="liste_ville">
        <option value="Paris">Paris</option>
        <option value="Lyon">Lyon</option>
        <option value="Marseille">Marseille</option>
        <option value="Lille">Lille</option>
        <option value="Strasbourg">Strasbourg</option>
      </select>      </td>
    </tr>
    <tr>
      <td><input type="button" name="Submit" value="S&eacute;lectionner"
onClick="maniplistemultiple()"></td>
    </tr>
  </table>
</form>
</body>
</html>
```

Le script s'exécute après un clic sur le bouton **Sélectionner**. La fonction maniplistemultiple débute en initialisant la variable ville. Puis, elle compte le nombre d'éléments présents dans la liste et transfère dans une variable nommée i, le numéro d'indice de la valeur sélectionnée. Ensuite, une boucle parcourt l'ensemble des éléments de la liste pour les afficher successivement dans une boîte de dialogue. Dans le cas où aucune ville n'est sélectionnée, rien ne s'affiche.

7. L'envoi de fichier par formulaire

C'est un type d'objet formulaire qui permet d'envoyer un fichier sur un serveur. Il se présente sous la forme d'un champ, de type texte, destiné à recevoir le chemin du fichier, placé sur le disque dur. Il est également possible d'utiliser le bouton **Parcourir** afin d'avoir un contrôle plus visuel. Attention cet objet peut être dangereux, car il est possible d'ajouter n'importe quel type de fichier y compris ceux comportant du code malveillant. Pour vérifier qu'il ne s'agit pas de fichier dangereux (c'est-à-dire tous les fichiers portant une autre extension que gif ou jpg, par exemple), il est préférable d'associer une fonction de contrôle. Pour effectuer ce genre de contrôle, il faut manipuler l'objet String. Un exemple de ce genre de contrôle est, donc, donné au chapitre Les principaux objets JavaScript en détail sur les chaînes de texte.

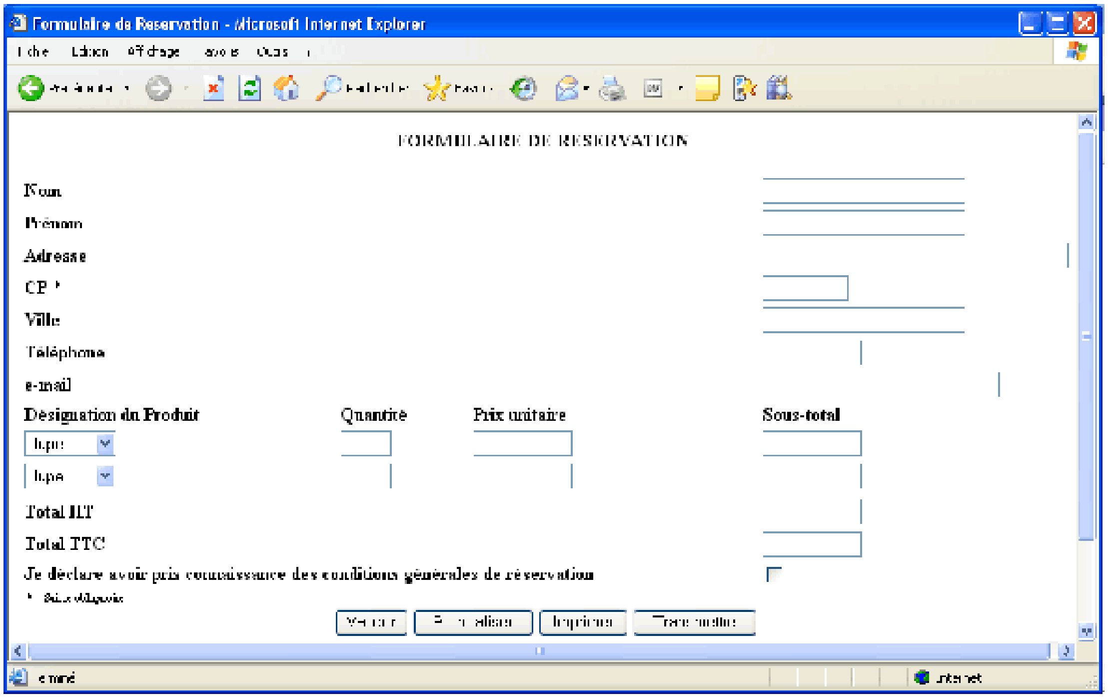
Valider et envoyer un formulaire par e-mail

Pour tout développeur de site web, les formulaires constituent un élément essentiel de l'interactivité avec les internautes. Ils permettent de recueillir des informations dans une base de données ou si l'usage d'une base n'est pas envisagé, de transmettre les informations saisies dans le formulaire par e-mail simplement.

Dans cet exemple, il s'agit de concevoir un formulaire de réservation d'articles qui pourra être imprimé et envoyé à une adresse e-mail déterminée. Les informations ne seront pas stockées dans une base de type MySql, par exemple, mais seront envoyées par e-mail à un destinataire prédéfini, qui pourra traiter ces données, dès réception.

La création du formulaire ne sera pas traitée dans cet ouvrage, il est possible de le créer en html ou par un éditeur HTML de type Dreamweaver, par exemple.

Comme précédemment, l'insertion du code JavaScript permettra de vérifier la présence d'informations dans les champs de formulaire et de contrôler leur contenu. Le résultat de ces contrôles permettra d'envoyer des messages d'avertissement ou encore d'effectuer des calculs.



Le formulaire comporte, en plus des éléments déjà détaillés, quatre boutons aux fonctions distinctes.

Le bouton **Transmettre** effectue la transmission des informations par e-mail.

Code du bouton :

```
<input name="bouton_transmettre" type="submit"
id="bouton_transmettre" value="Transmettre">
```

Ce bouton de type Submit déclenche l'envoi du formulaire.

Code du formulaire :

```
<form action="mailto:monmail@monserveur.com" method="post"
enctype="text/plain" name="Formulaire_reservation"
```

Le bouton **Réinitialiser** permet d'effacer le contenu de tous les contrôles du formulaire.

```
<input name="bouton_reinitialiser" type="reset"
id="bouton_reinitialiser" value="R&eacute;initialiser">
```

Le bouton **Imprimer** permet l'impression du formulaire.

```
function Imprim() {
window.print();}
```

Le bouton **Valider** permettra le déclenchement des différents traitements inclus dans la fonction JavaScript de contrôle.

Le premier traitement concerne le champ Nom. Pour ce champ, vous désirez convertir la saisie de minuscule en majuscule. Pour cela, vous ferez appel à la méthode toUpperCase() qui permet de convertir la chaîne de caractères.

Le second traitement teste le contenu du champ Code_Postal. Tout d'abord, vous effectuez un test pour interdire la possibilité d'un champ vide. Ensuite, vous le testez à nouveau pour n'autoriser que la saisie d'un champ numérique. Dans un autre cas, comme par exemple la saisie de lettre dans ce champ, un avertissement s'affiche dans une boîte de dialogue. Vous utiliserez IsNaN *(Is Not a Number)* pour faire le test.

Le troisième traitement permet de tester la validité du champ e-mail par la présence d'une arobase dans la chaîne de caractères. Vous utilisez pour le faire la méthode search('@') qui renvoie dans une variable, la position de l'arobase dans la chaîne de caractères. Si cette position est négative, cela signifie qu'il n'existe pas d'arobase dans la chaîne. L'adresse n'est donc pas valide. Bien sûr, vous pouvez procéder à d'autres tests, notamment pour savoir si l'extension du nom de domaine est réaliste. Malgré tout, il est impossible d'exclure la saisie d'une adresse farfelue de type inconnu@sansserveur.com.

Le quatrième traitement de cette fonction permet d'afficher le prix d'un article à partir de sa sélection dans la liste déroulante. Pour ce faire, vous utilisez l'instruction switch qui permet de traiter l'ensemble des cas de sélection de la liste. En fonction de cette sélection, vous affichez le prix dans le champ Prix_unitaire_P1 ou P2 correspondant chacun au choix d'un premier et d'un second produit.

Le cinquième traitement permet le calcul des sous-totaux P1 ou P2 en effectuant le produit de la valeur des champs Quantité et Prix_unitaire P1 ou P2.

Le sixième traitement calcule le sous-total Hors Taxe par addition des champs sous_total P1 et sous_total P2 qui sont préalablement convertis en nombre par l'instruction `parseInt()`.

Le septième traitement calcule les montants TTC par produit des sous-totaux avec 1.196 et arrondi en utilisant `Math.round`.

Le huitième traitement contrôle si la case à cocher **Réservation** est cochée. Si ce n'est pas le cas, une boîte de dialogue, rappelant cette obligation, s'affiche.

Le neuvième traitement permet d'imprimer le formulaire par l'instruction figurant dans la fonction Imprim(). Celle-ci va se déclencher lors du clic sur le bouton **Imprimer** du formulaire.

Pour terminer, vous transmettez les valeurs saisies ou calculées dans les champs du formulaire, en cliquant sur le bouton **Transmettre**.

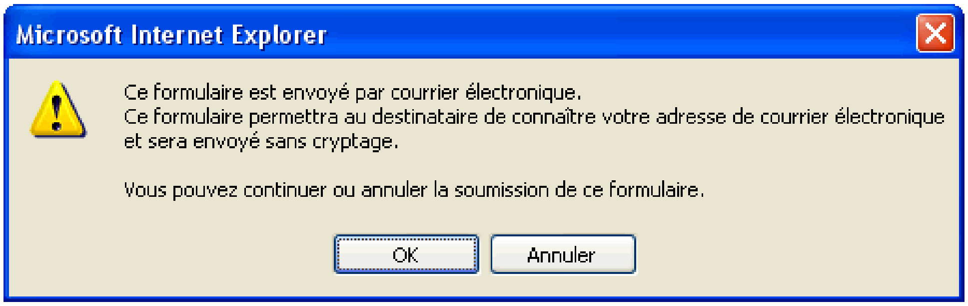
Il convient de noter que les informations transmises seront précédées du nom du champ défini, lors de la création du formulaire, et du signe égal (=).

Voici un exemple du contenu du e-mail reçu, après transmission du formulaire :

```
Nom=DURANT
Prenom=pascal
Adresse=12 rue des fleurs
CP=57000
Ville=METZ
Telephone=0387999999
email=mon_adresse@mon_serveur.com
Designation_P1=Jupe
Quantite_P1=5
Prix_unitaire_P1=35
Sous_total_P1=175
Designation_P2=Short
Quantite_P2=3
Prix_unitaire_P2=25
Sous_total_P2=75
Total_HT=250
Total_TTC=299
Reservation=checkbox
Submit4=Transmettre
```

Le bouton **Ré-initialiser** permet de mettre à zéro tous les champs du formulaire.

La transmission d'un formulaire par e-mail peut poser certains problèmes. Il est nécessaire d'ouvrir le client de messagerie par défaut et de faire un « envoyer-transmettre » pour terminer l'envoi.




```
<script language="JavaScript">
function control()
{
nom_minuscule=Formulaire_reservation.Nom.value;
nom_majuscule=nom_minuscule.toUpperCase();
Formulaire_reservation.Nom.value=nom_majuscule;
if(Formulaire_reservation.CP.value==’’) {
Formulaire_reservation.CP.style.backgroundColor = "FFCC00";
alert("La saisie de ce champ est obligatoire");
}
code=Formulaire_reservation.CP.value;
if(isNaN(code)) {
alert("Veuillez entrer un Code Postal valide");
Formulaire_reservation.CP.style.backgroundColor = "FFCC00";
Formulaire_reservation.CP.value=’’;
}
test_mail=Formulaire_reservation.email.value;
resultat = test_mail.search(’@’);
if (resultat<0) {
alert("e-mail non valide");
Formulaire_reservation.email.style.backgroundColor = "FFCC00";
Formulaire_reservation.email.value=’’
}
var article1 = Formulaire_reservation.Designation_P1.value;
var article2 = Formulaire_reservation.Designation_P2.value;
switch (article1)
{
case "Jupe":
Formulaire_reservation.Prix_unitaire_P1.value=35;
break;
case "Pantalon":
Formulaire_reservation.Prix_unitaire_P1.value=50;
break;
case "Short":
Formulaire_reservation.Prix_unitaire_P1.value=25;
break;
case "Tee-shirt":
Formulaire_reservation.Prix_unitaire_P1.value=15;
}
switch (article2)
{
case "Jupe":
Formulaire_reservation.Prix_unitaire_P2.value=35;
break;
case "Pantalon":
Formulaire_reservation.Prix_unitaire_P2.value=50;
break;
case "Short":
Formulaire_reservation.Prix_unitaire_P2.value=25;
break;
case "Tee-shirt":
Formulaire_reservation.Prix_unitaire_P2.value=15;
}
Formulaire_reservation.Sous_total_P1.value=Formulaire_reservation.
Quantite_P1.value*Formulaire_reservation.Prix_unitaire_P1.value;
Formulaire_reservation.Sous_total_P2.value=Formulaire_reservation.
Quantite_P2.value*Formulaire_reservation.Prix_unitaire_P2.value;
```

```
sous_total_P1=parseInt(Formulaire_reservation.Sous_total_P1.value);
sous_total_P2=parseInt(Formulaire_reservation.Sous_total_P2.value);
Total_HT=sous_total_P1+sous_total_P2;
Formulaire_reservation.Total_HT.value=Total_HT;
Total_TTC=Total_HT*1.196;
Total_TTC_arrondi=Math.round(Total_TTC*100)/100;
Formulaire_reservation.Total_TTC.value=Total_TTC_arrondi;
if (Formulaire_reservation.Reservation.checked==false)
alert("Veuillez valider les conditions générale de réservation");
}
```

La méthode round() de l'objet Math est traitée dans le chapitre Les principaux objets JavaScript en détail.

Pour valider les masques de saisie, il est également possible d'utiliser l'objet RegExp dans une expression régulière qui sera également traitée au chapitre Les principaux objets JavaScript en détail.

Le formulaire est un des objets important de la composition d'une page, cependant il y en a d'autres en JavaScript qu'il convient de bien décrire.

Objets et navigateurs

Comme nous l'avons vu au cours des chapitres précédents, le JavaScript est interprété de manière différente selon les navigateurs. Au sommet de la hiérarchie des objets JavaScript, il faut distinguer l'objet navigator qui correspond au navigateur utilisé pour la lecture de la page de l'objet window représentant la fenêtre.

Les objets JavaScript

JavaScript fournit un certain nombre d'objets définis dans le noyau et donc, manipulables à tout moment. Ces objets du noyau ne dépendent pas du navigateur avec lequel ils sont utilisés et ils seront détaillés en fin de chapitre.

1. L'objet navigator

L'objet navigator correspond au navigateur utilisé par le visiteur de la page. Cette information peut être primordiale notamment lorsque vous associez JavaScript et DHTML, l'interprétation de celui-ci étant différente selon les navigateurs et leur version. Bien évidemment, les informations de l'objet navigator sont en lecture seule, ce qui semble assez logique. Dans la hiérarchie des objets JavaScript, l'objet navigator est lié à l'objet window. Vous pouvez donc y accéder avec les deux syntaxes suivantes possibles :

`window.navigator` ou simplement par `navigator`.

Les propriétés de l'objet navigator permettent d'obtenir des informations essentielles pour orienter le visiteur et ainsi, lui permettre d'optimiser sa visite. Ainsi, le fait de savoir si les cookies sont activés ou pas permet d'alerter le visiteur, que le site ne pourra pas fonctionner correctement s'il ne les active pas. De même, la connaissance de la langue utilisée par le navigateur permettra d'orienter automatiquement le visiteur vers les pages correspondant à celle-ci.

a. Les propriétés

Propriété	Résultat
`appCodeName`	Retourne le code du navigateur.
`appName`	Retourne le nom du navigateur.
`appVersion`	Retourne la version du navigateur.
`cookieEnabled`	Retourne true ou false pour indiquer si les cookies sont activés.
`cpuClass`	Retourne le type de processeur de l'ordinateur utilisé.
`javaEnabled()`	Détermine si le navigateur est apte à exécuter des applets Java.
`mimeTypes`	Retourne un tableau des types mimes supportés par le navigateur.
`platform`	Retourne la plate-forme sur laquelle le navigateur fonctionne.
`plugins`	Retourne un tableau des plug-ins installés sur le poste du navigateur.
`userAgent`	Retourne toutes les informations concernant le navigateur client.
`userLanguage`	Retourne la langue utilisée par le navigateur.

Exemple : afficher les informations du navigateur utilisé grâce à la propriété userAgent.




```
<script language="JavaScript">
agent=navigator.userAgent;
alert(agent);
</script>
```

Les informations affichées correspondent au type et à la version du navigateur, à la plate-forme utilisée et à la version de CLR présente sur la machine. Le CLR correspond à la machine virtuelle du framework.NET de Microsoft.

b. Les méthodes

Les méthodes sont moins nombreuses et moins utiles. Elles permettent essentiellement de manipuler les préférences du navigateur.

Méthode	Résultat
`plugins.refresh()`	Rafraîchit la liste des plug-ins installés sur le poste du navigateur.
`preference()`	Détermine les préférences du navigateur.
`savePreferences()`	Sauvegarde les modifications apportées aux préférences du navigateur.

Exemple : afficher le nom du navigateur, sa version, le nom du type de plate-forme actuellement utilisé ainsi que la présence et l'activation des cookies, dans une boîte de dialogue.




```
<script language="JavaScript">
navigateur=navigator.appName;
version=navigator.appVersion;
plateforme=navigator.platform;
cookie=navigator.cookieEnabled();
if (cookie==true){
alert("Vous utilisez actuellement " +navigateur+ " "+version+ "\r
comme navigateur Internet, sur une plate-forme de type :
" +plateforme+ " avec les cookies activés" );
}
else
{
alert("Vous utilisez actuellement " +navigateur+ " "+version+ "\r
comme navigateur Internet, sur une plate-forme de type :
" +plateforme+ " Attention ! les cookies ne sont pas activés" );
}
</script>
```

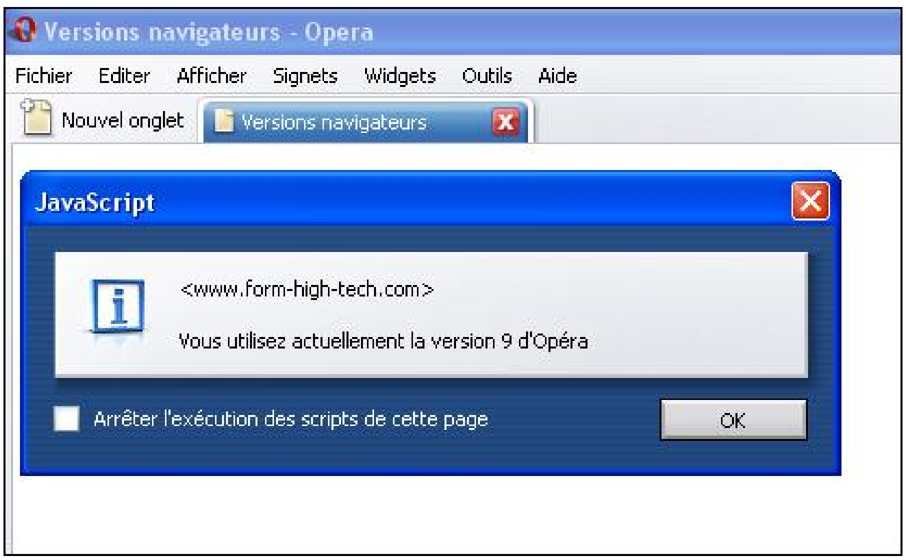
Le script débute par l'affectation de plusieurs variables par les propriétés et méthodes de l'objet navigator et notamment de cookieEnabled(), qui sert de valeur de comparaison du test. En testant cookieEnabled(), le script est orienté vers l'un ou l'autre des messages.

En ouvrant ce script avec Mozilla, Firefox, la réponse sera « Netscape ».

<u>*Exemple : afficher le type de navigateur et sa version pour permettre d'orienter le script ultérieurement.*</u>








```
<html>
<head>
<title>Verification du navigateur</title>
<script language="JavaScript">
function verifnavigateur() {
navigateur = navigator.appName.substring(0,3);
version = navigator.appVersion.substring(0,1);
if (navigateur == "Mic"){
alert("Vous utilisez actuellement la version " +version+" d'Internet
Explorer");
}
else if(navigateur=="Net") {
alert("Vous utilisez actuellement la version " +version+" de Firefox ");
}
else if(navigateur=="Ope") {
alert("Vous utilisez actuellement la version " +version+" d'Opéra ");
}
}
</script>
</head>
<body onLoad="verifnavigateur()">
</center>
</body>
</html>
```

Le script s'exécute au chargement de la page et utilise les propriétés de l'objet navigator pour donner une valeur aux variables navigateur et version. La valeur affectée correspond à une chaîne de trois caractères extraits de ces propriétés, par l'emploi de la méthode substring() de l'objet String que nous détaillerons ultérieurement dans ce chapitre. Il reste, ensuite, à comparer cette chaîne de caractères, au début du nom des navigateurs (ici Mic pour Microsoft Internet Explorer, Net pour Firefox Mozilla et Ope pour Opéra). Une boîte de dialogue peut, alors, être affichée en reprenant le contenu des variables.

2. L'objet window

L'objet window (fenêtre) est l'objet le plus élevé dans la hiérarchie des objets JavaScript. C'est le parent de tous les objets placés à l'intérieur. Cet objet est souvent qualifié d'implicite, car il n'est pas nécessaire de le nommer pour accéder aux objets placés en dessous. Cette simplicité d'utilisation est atténuée par le fait que, quelques-unes de ses propriétés et méthodes ne sont pas interprétées correctement par tous les navigateurs. Malgré tout, c'est un objet qui est souvent utilisé, car c'est celui qui comporte le plus grand nombre de propriétés et de méthodes. Il est donc très important de les détailler.

a. Les propriétés

Propriété	Résultat	Reconnu par :
`closed`	Retourne true si la fenêtre à laquelle on se réfère est fermée.	Internet Explorer, Mozilla, Firefox, Opéra.
`defaultStatus`	Correspond au message permanent affiché dans la barre de statuts située en dessous de la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`document`	Correspond au document courant.	Internet Explorer, Mozilla, Firefox, Opéra.
`frames`	Correspond à l'ensemble des cadres de l'objet window.	Internet Explorer, Mozilla, Firefox, Opéra.
`history`	Correspond au contenu de l'objet history.	Internet Explorer, Mozilla, Firefox, Opéra.
`innerHeight`	Correspond à la hauteur utilisable d'une fenêtre.	Mozilla, Firefox, Opéra.
`innerWidth`	Correspond à la largeur utilisable d'une fenêtre.	Mozilla, Firefox, Opéra.
`length`	Correspond au nombre total de cadres utilisés dans la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`location`	Correspond à l'URL de la page chargée dans la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`locationbar`	Correspond à la barre d'adresse du navigateur.	Mozilla, Firefox, Opéra.
`menubar`	Correspond la barre de menu du navigateur.	Mozilla, Firefox, Opéra.
`name`	Correspond au nom donné à la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`opener`	Correspond au nom de la fenêtre qui a créé une fenêtre au moyen de la méthode Open().	Internet Explorer, Mozilla, Firefox, Opéra.

`outerHeight`	Correspond à la hauteur extérieure de la page en pixels.	Mozilla, Firefox, Opéra.
`outerWidth`	Correspond à la largeur extérieure de la page en pixels.	Mozilla, Firefox, Opéra.
`pageXoffset`	Correspond à la position horizontale en pixels de la fenêtre.	Mozilla, Firefox, Opéra.
`pageYoffset`	Correspond à la position verticale en pixels de la fenêtre.	Mozilla, Firefox, Opéra.
`parent`	Correspond à la page comprenant l'ensemble des cadres d'une page de cadres.	Internet Explorer, Mozilla, Firefox, Opéra.
`personalbar`	Correspond à la barre d'outils personnelle.	Mozilla, Firefox, Opéra.
`scrollbars`	Correspond aux barres de défilement horizontal et vertical.	Mozilla, Firefox, Opéra.
`self`	Correspond à la fenêtre courante (window).	Internet Explorer, Mozilla, Firefox, Opéra.
`status`	Correspond au message aléatoire qui peut être affiché lors d'un évènement dans la barre de statuts située en dessous de la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`toolbar`	Correspond à la barre d'outils du navigateur.	Mozilla, Firefox, Opéra.
`top`	Correspond à la fenêtre du plus haut niveau à l'intérieur d'une page de cadres.	Internet Explorer, Mozilla, Firefox, Opéra.
`window`	Correspond à la fenêtre courante (self).	Internet Explorer, Mozilla, Firefox, Opéra.

b. Les méthodes

Certaines d'entres elles ont déjà été traitées au chapitre Utilisation des constantes, variables et opérateurs. Pour le reste, il s'agit de méthodes concernant la gestion des fenêtres (ouverture, fermeture, position, affichage, etc.), ou la navigation des pages visitées. Le tableau suivant fournit un descriptif de l'ensemble des méthodes de l'objet window :

Méthode	Résultat	Reconnu par
`alert()`	Affiche une boîte de dialogue comportant un message d'avertissement et un bouton **OK**.	Internet Explorer, Mozilla, Firefox, Opéra.
`back()`	Revient d'une page en arrière dans l'historique des pages visitées.	Mozilla, Firefox, Opéra.
`blur()`	Désactive la fenêtre courante.	Internet Explorer, Mozilla, Firefox, Opéra.
`close()`	Ferme la fenêtre courante.	Internet Explorer, Mozilla, Firefox, Opéra.

`confirm()`	Affiche une boîte de dialogue comportant deux boutons : **OK** et **Annuler**.	Internet Explorer, Mozilla, Firefox, Opéra.
`find()`	Effectue une recherche de texte dans la page active.	Mozilla, Firefox, Opéra.
`focus()`	Active une fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`forward()`	Avance d'une page en avant dans l'historique des pages visitées.	Mozilla, Firefox, Opéra.
`home()`	Charge la page définie comme page d'accueil dans le navigateur.	Mozilla, Firefox, Opéra.
`moveTo()`	Déplace la fenêtre active.	Internet Explorer, Mozilla, Firefox, Opéra.
`open()`	Ouvre une nouvelle fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`print()`	Imprime la page active.	Internet Explorer, Mozilla, Firefox, Opéra.
`prompt()`	Affiche une boîte de dialogue permettant à l'utilisateur de saisir une valeur.	Internet Explorer, Mozilla, Firefox, Opéra.
`resizeBy()`	Modifie la taille de la fenêtre active à partir du coin inférieur droit en fonction d'une quantité de pixels indiquée.	Internet Explorer, Mozilla, Firefox, Opéra.
`resizeTo()`	Modifie la taille de la fenêtre active en déterminant la position du coin inférieur droit.	Internet Explorer, Mozilla, Firefox, Opéra.
`setInterval()`	Effectue un traitement à intervalle régulier.	Internet Explorer, Mozilla, Firefox, Opéra.
`setTimeOut()`	Déclenche une minuterie.	Internet Explorer, Mozilla, Firefox, Opéra.
`scrollBy()`	Décale le contenu d'une fenêtre en fonction d'une quantité exprimée en pixels.	Internet Explorer, Mozilla, Firefox, Opéra.
`scrollTo()`	Décale le contenu d'une fenêtre en déterminant la nouvelle origine du coin supérieur gauche.	Internet Explorer, Mozilla, Firefox, Opéra.
`stop()`	Interrompt le chargement de la page actuelle.	Internet Explorer, Mozilla, Firefox, Opéra.

Une des méthodes de l'objet window doit être détaillée car elle est fréquemment employée.

c. Méthode open()

Cette méthode est souvent employée pour créer ce que l'on appelle des pop-up, c'est-à-dire des fenêtres qui s'ouvrent automatiquement pendant que vous consultez une page. Quelquefois utiles, quelquefois embarrassantes, ces fenêtres surgissantes peuvent être bloquées par un anti pop-up, qui est d'ailleurs inclus dans les dernières versions des navigateurs. C'est pour cette raison qu'il ne faut jamais inclure un contenu important pour votre site (comme un formulaire d'inscription, par exemple), dans un pop-up. Le mieux consiste à réserver l'usage des pop-up à

des messages de rappel (date et heure de réunion, par exemple) ou pour des informations se renouvelant rapidement.

La syntaxe de la méthode open() est la suivante :

```
f=window.open("page", "nom", "paramètre1,parametre2,parametre3")
```

Où f est le nom de l'objet représenté par la nouvelle fenêtre, page est l'adresse de la page à afficher, nom est le nom de la nouvelle fenêtre et paramètre 1/2/3... correspondent à des paramètres optionnels (position, taille, etc.). Les deux premiers paramètres sont obligatoires, les autres sont facultatifs. Les paramètres sont indiqués sous la forme d'une chaîne de caractères, sans limite de taille. Attention à certains paramètres, qui ne sont pas interprétés par Internet Explorer. Seuls les paramètres de position et de taille sont exprimés numériquement, les autres utilisent les valeurs (true ou false) ou (yes ou no). Le tableau suivant précise les différents paramètres possibles et leur reconnaissance par les navigateurs.

Paramètre	**Fonction**	**Reconnu par**
`alwaysLowered`	Crée une nouvelle fenêtre sous les autres.	Mozilla, Firefox, Opéra..
`alwaysRaised`	Crée une nouvelle fenêtre par-dessus toutes les autres.	Mozilla, Firefox, Opéra.
`dependent`	Crée une nouvelle fenêtre dépendante de la première. En cas de fermeture de la fenêtre parente, la fenêtre fille sera fermée également.	Mozilla, Firefox, Opéra..
`directories`	Affiche la barre d'outils personnelle.	Internet Explorer, Mozilla, Firefox, Opéra.
`focus`	Donne le focus clavier. Permet ainsi de réactiver une fenêtre qui vient d'être créée.	Internet Explorer, Mozilla, Firefox, Opéra.
`fullscreen`	Permet d'afficher la nouvelle fenêtre en plein écran.	Internet Explorer
`height`	Définie en pixels la hauteur de la nouvelle fenêtre. Doit être remplacée par innerHeight sur Internet Explorer.	Internet Explorer, Mozilla, Firefox, Opéra.
`hist`	Détermine si l'URL de la nouvelle fenêtre sera stockée dans l'historique du navigateur.	Internet Explorer, Mozilla, Firefox, Opéra.
`hotkeys`	Active ou désactive certains raccourcis-clavier pour la nouvelle page.	Mozilla, Firefox, Opéra.
`innerHeight`	Détermine la hauteur en pixels de la nouvelle fenêtre (celle-ci doit être supérieure à 100 pixels).	Mozilla, Firefox, Opéra.
`innerWidth`	Détermine la largeur en pixels de la nouvelle fenêtre (celle-ci doit être supérieure à 100 pixels).	Mozilla, Firefox, Opéra.
`left`	Détermine la position en abscisse de la nouvelle fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`location`	Affiche ou masque la barre d'adresse.	Internet Explorer, Mozilla, Firefox, Opéra.

`menubar`	Affiche ou masque la barre de menus située en haut de la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`outerHeight`	Détermine en pixels la hauteur du cadre extérieur (celle-ci doit être supérieure à 100 pixels).	Mozilla, Firefox, Opéra.
`outerWidth`	Détermine en pixels la largeur du cadre extérieur (celle-ci doit être supérieure à 100 pixels).	Mozilla, Firefox, Opéra.
`resizable`	Permet la modification de la taille de la nouvelle fenêtre par l'utilisateur.	Internet Explorer, Mozilla, Firefox, Opéra.
`screenX`	Détermine la position en abscisse du coin supérieur gauche de la nouvelle fenêtre.	Mozilla, Firefox, Opéra.
`screenY`	Détermine la position en ordonnée de la nouvelle fenêtre.	Mozilla, Firefox, Opéra.
`scrollbars`	Affiche les barres de défilement.	Internet Explorer, Mozilla, Firefox, Opéra.
`status`	Affiche ou masque la barre de statuts située au bas de la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.
`toolbar`	Affiche ou masque la barre d'icônes située au-dessous de la barre de menus.	Internet Explorer, Mozilla, Firefox, Opéra.
`width`	Définit en pixels la largeur de la fenêtre.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher et masquer une nouvelle fenêtre à partir d'une autre.




```
<html>
<head>
<title>Objet Window</title>
<script language="javascript">
fenetre=window.open("","Nouvelle","height=100,width=300,menubar=yes,
```

```
toolbar=yes,resizable=no,scrollbar=no");
</script>
</head>
<body onUnload="window.fenetre.close()">
<input type="button" name="Submit" value="Nouvelle"
onClick="fenetre.focus()">
<input type="button" name="Submit2" value="Masquer"
onClick="fenetre.blur()">
</body>
</html>
```

Le script utilise les méthodes focus() et blur() de l'objet window pour afficher la fenêtre souhaitée. Ainsi, dès le début du script, une nouvelle fenêtre s'affiche avec les paramètres de taille et d'affichage souhaités. L'instruction <body onUnload="window.fenetre.close()"> permet de refermer la fenêtre, en cas de déchargement de la première fenêtre. Ensuite, les boutons **Nouvelle** et **Masquer** déclenchent le lancement des deux fonctions permettant l'affichage et le masquage de la fenêtre.

Exemple : modifier la taille et la position d'une fenêtre de manière répétée.

```
<html>
<head>
<head>
<title>Fenêtre modifiant la taille et la position</title>
<script language="JavaScript"  type="text/javascript">
function tailleposition(){
window.innerHeight=100;
window.innerWidth=200;
for(i=0;i<200;i++){
h=4;
z=4;
self.moveBy(h, z);
self.resizeBy(h, z);
}
}
</script>
</head>
<body>
<form name="form1" action="">
  <input name="Modifier" type="button" id="Modifier"
onClick='tailleposition();' value="Modifier">
</form>
</body>
</html>
```

Dans ce script, les propriétés de taille et de position sont utilisées. Le script débute par la définition de départ de la fenêtre, avec les propriétés innerHeight et innerWidth. Ensuite, une boucle est créée permettant de redimensionner la fenêtre et de la faire bouger de quatre pixels supplémentaires (la fenêtre se déplacera donc du bord supérieur gauche au bord inférieur droit, ou donne l'impression de vibrer si la fenêtre prend déjà toute la place de l'écran). Une fois la boucle terminée, la fenêtre se stabilise.

Exemple : proposer d'insérer la page active en favoris :





```
<html>
<head>
<title>Ajout en favoris</title>
<script language="JavaScript">
navigateur = navigator.appName.substring(0,3);
version = navigator.appVersion.substring(0,1);
adresse=window.location;
titre=window.name;
function ajoutfavoris() {
rep=confirm("Voulez-vous ajouter cette page dans vos favoris ?");
if (rep==false) {
return;
}
else {
if (navigateur == "Mic" && version >= 4){
url_site=adresse;
titre_site = titre;
window.external.AddFavorite(url_site, titre_site);
}
else {
alert("Utilisez le raccourci-clavier Ctrl+D pour ajouter cette page
à vos favoris");
}
}
}
</script>
</head>
<body onLoad="ajoutfavoris()">
</center>
</body>
</html>
```

Ce script s'exécute au chargement de la page et le premier traitement effectué par la fonction ajoutfavoris() est de récupérer le nom de la page, le nom et la version du navigateur utilisé, car le traitement de mise en favoris est différent selon les cas. Avec Internet Explorer, cet ajout peut s'effectuer automatiquement, mais avec Firefox/Mozilla, Firefox il ne peut être réalisé qu'à partir d'un raccourci-clavier ou au choix de l'utilisateur dans le menu, et ce pour des raisons de sécurité. Ensuite, il est demandé à l'utilisateur s'il souhaite ou non ajouter cette page. S'il ne le désire pas, le script est redirigé par l'instruction `return` et l'utilisateur poursuit sa visite. Dans le cas contraire, il faut distinguer la version du navigateur. C'est la raison pour laquelle, le script teste les trois premiers caractères de la variable correspondant au nom du navigateur et si celle-ci correspond à la chaîne « Mic », par exemple, il est possible d'en déduire qu'il s'agit d'Internet Explorer de Microsoft. Il est alors possible d'ajouter la page en favoris par l'instruction window.external.AddFavorite.

Exemple : afficher dans une nouvelle fenêtre appelée ENI, sans barre d'adresse, la page http://www.editions-eni.fr après avoir cliqué sur un bouton situé sur la page.

```
<html>
<head>
<title>L'objet window</title>
<script language="JavaScript">
function popup() {
f=window.open("http://www.editions-eni.fr/","ENI","location=no");
}
</script>
</head>
<body>
<input type="submit" name="Submit" value="Popup" onclick="popup()" />
</body>
</html>
```

Pour tester ce script, il vous faut désactiver les bloqueurs de pop-up de votre navigateur. Pour rappel, certains paramètres ne sont pas reconnus par Internet Explorer.

Exemple : afficher la page http://www.editions-eni.fr dans une nouvelle fenêtre appelée ENI, sans barre d'adresse, simplement par chargement de la page.

```
<body>
<body onload=popup();
</body>
```

Il suffit d'appeler la fonction pop-up directement dans le corps de la page par l'instruction <body onload=popup();.

Exemple : afficher la page http://www.editions-eni.fr dans une nouvelle fenêtre appelée ENI, en mode plein écran (impossible dans Mozilla, Firefox ou Opéra.).

```
f=window.open("http://www.editions-eni.fr/","ENI","fullscreen=yes");
```

d. Méthode Close()

Elle permet de fermer les fenêtres ouvertes par la méthode Open(), en utilisant le nom employé lors de sa création. La syntaxe de la méthode Close() est la suivante :

```
Nomdelafenêtreàfermer.Close() ;
```

Exemple : créer un bouton permettant d'afficher la page http://www.editions-eni.fr dans une nouvelle fenêtre appelée f, un autre bouton permettant de fermer cette même fenêtre et un dernier permettant de fermer la fenêtre active.




```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<title>L'objet window - Méthode Close</title>
<script language="JavaScript">
function popup() {
f=window.open("http://www.editions-eni.fr/","ENI", width=200,
height=200,"location=no");
}
function fermerf() {
f.close();
}
function fermeractive(){
self.close();
}
</script>
</head>
<body>
<p>
```

```
  <input type="button" name="Submit" value="Popup" onClick="popup()" />
</p>
<p>
  <input type="button" name="Submit2" value="Fermer fenetre f"
onClick="fermerf()" />
</p>
<p>
  <input type="button" name="Submit3" value="Fermer fenetre active"
onClick="fermeractive()">
</p>
</body>
</html>
```

Le script se décompose en trois fonctions distinctes. La première permet d'afficher la nouvelle fenêtre en chargeant la page désirée. La seconde ferme la nouvelle fenêtre et la troisième ferme la fenêtre de démarrage (c'est-à-dire celle où se situe le script, l'emploi du mot clé self permet de la désigner).

> Attention ! La méthode close() ne fonctionne pas avec Mozilla/Firefox.

Exemple : orienter le visiteur vers la page Internet de son navigateur (Mozilla, Firefox, Internet Explorer ou Opéra) en fonction de celui-ci.



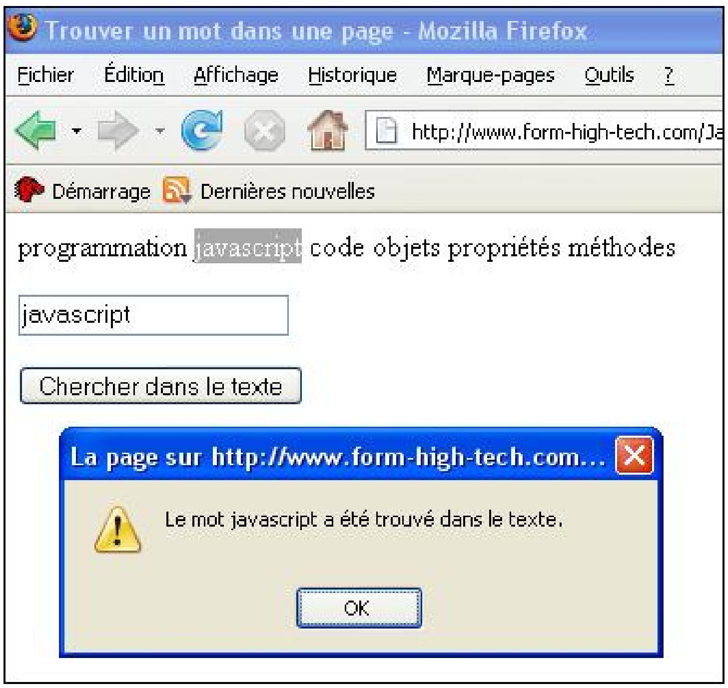



```
<html>
<head>
<title>redirection</title>
<script language="JavaScript">
function redirection() {
alert(navigator.appName);
if (navigator.appName=="Netscape") {
```

```
window.location=" http://firefox.fr/";
}
else if(navigator.appName=="Microsoft Internet Explorer") {
window.location="http://www.microsoft.com/france/windows/ie/";
}
else if(navigator.appName=="Opera") {
window.location=" http://www.opera.com/";
}
else {
window.location=" http://www.editions-eni.fr ";
}
}
</script>
</head>
<body>
<body onload="redirection()">
</body>
</html>
```

Le script se base sur une série de tests imbriqués et de ré-orientation à l’aide de window.location, qui permet de charger la page de présentation du navigateur utilisé. L’ensemble de ces instructions constitue une fonction, qui est exécutée au moment du chargement de la page. Dans le cas où le navigateur ne fait pas partie de la liste (Safari, Konqueror), la page à afficher correspond à celle des Éditions ENI.

Exemple : chercher un mot saisi, dans un champ de formulaire, à l’intérieur d’un texte et le mettre en surbrillance lorsqu’il est trouvé.




```
<html>
<head>
<title>Trouver un mot dans une page</title>
<script language="javascript">
var texte="programmation javascript code objets propriétés méthodes";
document.write(texte);
function chercher(critere) {
var trouve=find(critere,false,false);
if(trouve==true) {
alert("Le mot "+critere+" a été trouvé dans le texte.");
}
else {
alert("Le mot "+critere+"n’a pas été trouvé dans le texte");
}
}
</script>
</head>
<body>
```

```
<form id="form1" name="form1" method="post" action="">
  <p>
    <input name="nom" type="text" id="texte" value="" />
</p>
  <p>
    <input type="button" value="Chercher dans le texte"
onclick="chercher(form1.nom.value, false)" />
</p>
</form>
</body>
</html>
```

Ici la méthode find() de l'objet Window est employée. Sa syntaxe est la suivante :

```
find(chaînedecaractères, casse,sens) ;
```

Le paramètre casse prend la valeur true, si la recherche doit s'effectuer en distinguant les majuscules des minuscules et le paramètre sens prend la valeur true, si la recherche doit s'effectuer en débutant par la fin du texte.

Ici, la valeur, saisie dans le champ de formulaire, est utilisée dans la fonction chercher comme le paramètre critère de la méthode find(). Si le résultat de la méthode renvoie true, cela signifie que le mot a été trouvé en fonction des options de paramètres passées. Il faut alors afficher le résultat de réussite ou d'échec dans une boîte de dialogue.

La mise en surbrillance est faite automatiquement par le navigateur.

Attention ! La méthode find() n'est pas supportée par Internet Explorer.

3. L'objet document

L'objet document correspond à la page HTML elle-même. C'est un objet souvent manipulé en JavaScript notamment lorsqu'il est fait usage du DHTML. Par cet objet, il est possible d'accéder aux éléments composant la page, pour en modifier les propriétés ou le contenu. De même, par l'objet document, vous accédez aux cookies. Les propriétés et méthodes de l'objet document sont nombreuses :

a. Les propriétés

Propriété	Résultat	Reconnu par
`alinkColor`	Correspond à la couleur d'affichage des liens activés.	Internet Explorer, Mozilla, Firefox, Opéra.
`anchor`	Correspond au nom du tableau contenant toutes les ancres du document.	Internet Explorer, Mozilla, Firefox, Opéra.
`applets`	Correspond au nom du tableau contenant toutes les applets Java.	Internet Explorer, Mozilla, Firefox, Opéra.
`bgColor`	Correspond à la couleur de fond du document.	Internet Explorer, Mozilla, Firefox, Opéra.
`contextual`	Correspond au style de l'objet désigné et contenu dans le document.	Internet Explorer, Mozilla, Firefox, Opéra.
`cookie`	Correspond à une chaîne de caractères placée par le navigateur sur le poste client.	Internet Explorer, Mozilla, Firefox, Opéra.

`domain`	Correspond au nom de domaine qui a fourni le document HTML.	Mozilla, Firefox, Opéra.
`embeds`	Correspond au nom du tableau contenant tous les objets incorporés.	Internet Explorer, Mozilla, Firefox, Opéra.
`fgColor`	Correspond à la couleur utilisée pour écrire dans le document HTML.	Internet Explorer, Mozilla, Firefox, Opéra.
`forms`	Correspond au nom du tableau contenant tous les formulaires du document.	Mozilla, Firefox, Opéra.
`images`	Correspond au nom du tableau contenant toutes les images du document.	Mozilla, Firefox, Opéra.
`lastModified`	Correspond à la date de dernière modification du document.	Internet Explorer, Mozilla, Firefox, Opéra.
`layers`	Correspond au nom du tableau contenant toutes les entrées pour les layers du document.	Internet Explorer, Mozilla, Firefox, Opéra.
`linkColor`	Correspond à la couleur des liens des pages non encore visitées.	Internet Explorer, Mozilla, Firefox, Opéra.
`links`	Correspond au nom du tableau contenant tous les appels de liens du document.	Mozilla, Firefox, Opéra.
`plugins`	Correspond au nom du tableau contenant toutes les références et appel de plug-ins.	Mozilla, Firefox, Opéra.
`referrer`	Correspond à l'URL de la page ayant permis le chargement de la page active.	Internet Explorer, Mozilla, Firefox, Opéra.
`title`	Correspond à une chaîne de caractères représentant le titre de la page HTML	Internet Explorer, Mozilla, Firefox, Opéra.
`URL`	Correspond à une chaîne de caractères représentant l'URL du document présent dans la page HTML.	Internet Explorer, Mozilla, Firefox, Opéra.
`vlinkColor`	Correspond à la couleur d'affichage des liens des pages visitées.	Internet Explorer, Mozilla, Firefox, Opéra.

b. Les méthodes

Méthode	Résultat	Reconnu par
`captureEvents()`	Détermine le nom de l'évènement pour lequel la capture est autorisée pour le document actif.	Mozilla, Firefox, Opéra.
`close()`	Referme le document actif.	Internet Explorer, Mozilla, Firefox, Opéra.

`getSelection()`	Renvoie une chaîne de caractères correspondant au texte sélectionné dans le document actif.	Internet Explorer, Mozilla, Firefox, Opéra.
`handleEvent()`	Active le gestionnaire de l’évènement spécifié.	Internet Explorer, Mozilla, Firefox, Opéra.
`home()`	Active et charge la page de démarrage	Internet Explorer
`open()`	Active un document.	Internet Explorer, Mozilla, Firefox, Opéra.
`releaseEvents()`	Donne le nom de l’évènement dont la capture est restituée par le document courant.	Mozilla, Firefox, Opéra.
`routeEvents()`	Passe l’évènement capturé à la hiérarchie normale des évènements.	Mozilla, Firefox, Opéra.
`write()`	Autorise l’écriture dans le document actif.	Internet Explorer, Mozilla, Firefox, Opéra.
`writeln()`	Identique à write() mais ajoute le caractère newline (\n), ce qui permet un retour à la ligne.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher la date de dernière modification d’une page HTML au chargement de la page.

Page modifiée le : 12/05/2008 à 14 h 12 mins 52 secondes

```
<html>
<head>
<title>Derniere version</title>
<script language="javascript">
function dernieremodif() {
datemaj=document.lastModified;
var datemodif = new Date(datemaj);
var mois=datemodif.getMonth()+1;
var jour=datemodif.getDay();
var annee=datemodif.getFullYear();
var heure=datemodif.getHours();
var minute=datemodif.getMinutes();
var secondes=datemodif.getSeconds();
document.write("Page modifiée le : "+jour+"/"+mois+"/"+annee+" à
"+heure+ " h "+minute+" mins "+secondes+" secondes");
}
</script>
</head>
<body onload="dernieremodif()">
</body>
</html>
```

Le script s’exécute au chargement de la page. La fonction dernieremodif() permet d’extraire la date de la dernière modification de la page et de la convertir, car, sinon, le résultat est renvoyé au format anglais. Après extraction de cette date de dernière modification, il faut créer une variable de type date appelée datemodif. Ensuite, il est possible d’extraire, l’année, le mois, le jour ainsi que l’heure, les minutes et les secondes de la dernière sauvegarde. Il ne reste plus qu’à élaborer la chaîne alternant texte et variables pour afficher la réponse dans le document.

Exemple : proposer d’ajouter le site des Éditions ENI en page de démarrage :

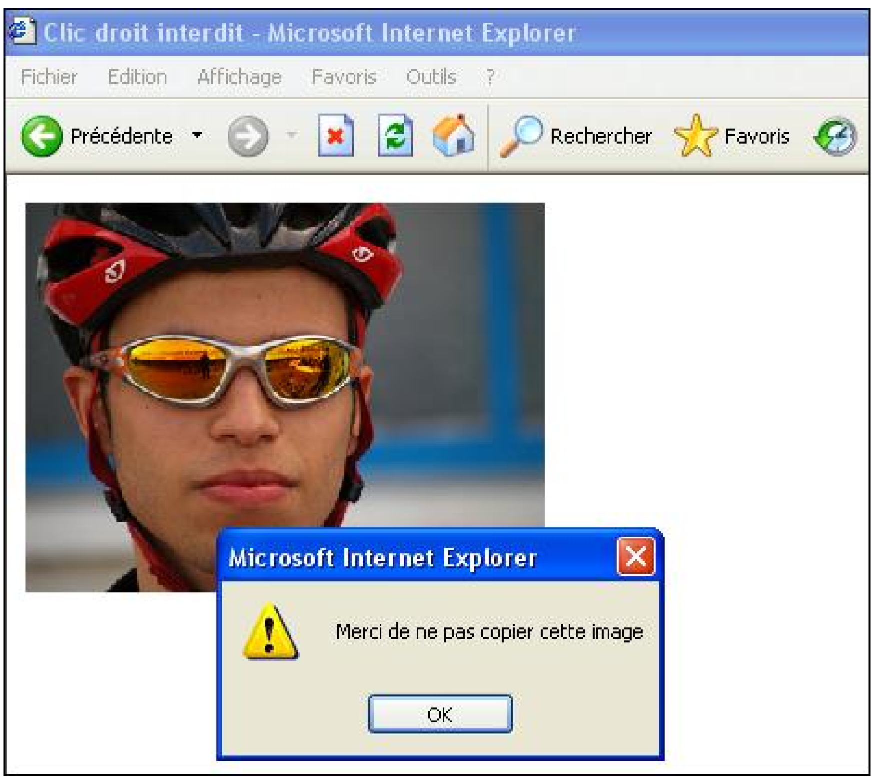



```
<html>
<head>
<title>Page en favoris</title>
</head>
<body>
<a href="#" onclick="this.style.behavior='url(#default#homepage )';
this.setHomePage('http://www.editions-eni.fr');"
title="Mettre le site editions-eni en favoris">Mettre le site
editions-eni en favoris</a>
</body>
</html>
```

Le script s'exécute au moment où l'utilisateur clique sur le lien figurant dans la page. Le script JavaScript indique alors au navigateur de modifier la page de démarrage, en prenant comme style une chaîne de type url. Vous noterez ici que le script JavaScript est directement inséré dans le code HTML.

Attention, ce script ne fonctionne qu'avec Internet Explorer et à condition que le niveau de sécurité soit au minimum. Il se peut tout de même que l'autorisation soit refusée par le navigateur.

Exemple : bloquer le clic droit souris pour informer le visiteur que la copie d'image est interdite :




```
<html>
<head>
<title>Clic droit interdit</title>
<script language="JavaScript"  type="text/javascript">
function copieinterdite(){
alert("Merci de ne pas copier cette image");
return(false);
```

```
}

</script>
</head>
<body onContextMenu = copieinterdite();>
<img src="velo.jpg" width="280" height="215">
</body>
</html>
```

Le script se base sur un évènement de l'objet document (document.onContextMenu). Ainsi, dès qu'un visiteur effectue un clic droit sur une image, une boîte de dialogue d'avertissement s'affiche.

Ce script ne prétend pas résoudre intégralement le problème de la copie d'image, c'est simplement un exemple à vocation pédagogique. Il est impossible de bloquer toutes les possibilités de copie (à partir de la touche printscreen, ou en enregistrant la page ou encore en maintenant la touche [Alt] enfoncée lors de la copie...).

4. L'objet screen

Il correspond à l'écran utilisé par le visiteur. Il ne possède pas de méthode et a six propriétés.

a. Les propriétés

Propriété	Résultat	Reconnu par
`availHeight`	Correspond à la hauteur de l'écran utilisé, en pixels.	Internet Explorer, Mozilla, Firefox, Opéra.
`availWidth`	Correspond à la hauteur de l'écran utilisé, en pixels.	Internet Explorer, Mozilla, Firefox, Opéra.
`colorDepth`	Correspond à la profondeur de couleurs de l'écran, c'est-à-dire au nombre de couleurs que celui-ci est capable de restituer.	Internet Explorer, Mozilla, Firefox, Opéra.
`height`	Correspond à la hauteur totale d'affichage de l'écran.	Internet Explorer, Mozilla, Firefox, Opéra.
`pixelDepth`	Correspond à la résolution de l'écran en nombre de couleurs, exprimées en bits par pixel.	Mozilla, Firefox, Opéra.
`width`	Correspond à la largeur d'affichage totale de l'écran.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher une boîte de dialogue dans laquelle apparaît la résolution utilisée en pixels ainsi que la profondeur de couleurs (8, 16 ou 32 bits).




```
<script language="javascript">
var alargeur=screen.availWidth;
var ahauteur=screen.availHeight;
var largeur =screen.width;
var hauteur=screen.height;
var couleurs=screen.colorDepth;
alert("La résolution de votre écran est de : "+largeur+" pixels de
largeur et de "+hauteur+" pixels en hauteur avec une palette de
```

```
couleurs de "+couleurs+" bits");
alert("La surface utile de votre affichage n'est que de "+alargeur+"
pixels de largeur et de "+ahauteur+" pixels en hauteur");
</script>
```

Le script utilise toutes les propriétés de l'objet screen pour définir les variables qui seront utilisées dans les messages affichés dans les boîtes de dialogue. La première permet de visualiser la résolution intégrale de l'écran, la seconde ne prend pas en compte les pixels en hauteur utilisés par la barre d'état du système d'exploitation. Les variables `screen.availHeight` et `screen.height` ont donc deux valeurs différentes.

5. L'objet string

Il convient de ne pas confondre le type de variable et l'objet string. En effet, l'objet string est un objet qui permet de manipuler les chaînes de caractères ; c'est pour cela qu'il est très souvent utilisé que ce soit pour la recherche, la concaténation ou l'extraction de sous-chaîne de caractères. Il n'est donc pas étonnant de constater que l'objet string dispose de nombreuses propriétés et méthodes.

a. Les propriétés

Propriété	Résultat	Reconnu par
`constructor`	Indique comment a été créé un objet référencé.	Internet Explorer, Mozilla, Firefox, Opéra.
`length`	Correspond à la longueur d'une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`prototype`	Permet d'ajouter des propriétés à un objet.	Internet Explorer, Mozilla, Firefox, Opéra.

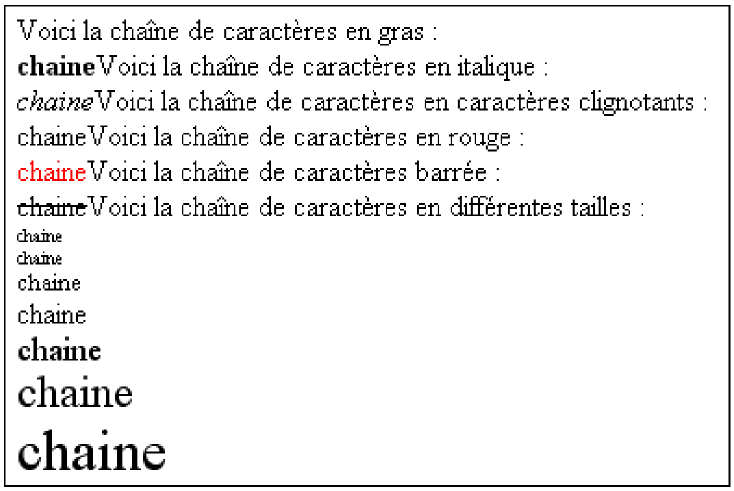
b. Les méthodes

Méthode	Résultat	Reconnu par
`anchor()`	Détermine une ancre nommée dans un document HTML.	Internet Explorer, Mozilla, Firefox, Opéra.
`big()`	Augmente d'un niveau la taille de la police utilisée dans un document HTML.	Internet Explorer, Mozilla, Firefox, Opéra.
`blink()`	Permet de faire clignoter le texte dans un document HTML.	Internet Explorer, Mozilla, Firefox, Opéra.
`bold()`	Affiche une chaîne de caractères en gras.	Internet Explorer, Mozilla, Firefox, Opéra.
`chartAt()`	Permet d'accéder à un seul caractère dans une chaîne complète.	Internet Explorer, Mozilla, Firefox, Opéra.
`charCodeAt()`	Renvoie la valeur en décimal du caractère indiqué en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`concat()`	Concatène plusieurs chaînes de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`eval()`	Convertit une chaîne de caractères en code de programme JavaScript.	Internet Explorer, Mozilla, Firefox, Opéra.

`fixed()`	Affiche une chaîne de caractères avec une police à pas fixe.	Internet Explorer, Mozilla, Firefox, Opéra.
`fontcolor()`	Modifie la couleur de la police de caractères d'une chaîne.	Internet Explorer, Mozilla, Firefox, Opéra.
`fontsize()`	Modifie la police de caractères d'une chaîne.	Internet Explorer, Mozilla, Firefox, Opéra.
`fromCharCode()`	Renvoie une chaîne de caractères constituée à partir de représentations isolées d'une suite de caractères Unicode.	Internet Explorer, Mozilla, Firefox, Opéra.
`indexof()`	Renvoie la position, à partir de la gauche, d'un caractère passé en argument dans une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`italics()`	Affiche une chaîne de caractères en italique.	Internet Explorer, Mozilla, Firefox, Opéra.
`lastIndexof()`	Renvoie la position, à partir de la droite, d'un caractère passé en argument dans une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`link()`	Demande le chargement d'une URL par le navigateur.	Internet Explorer, Mozilla, Firefox, Opéra.
`match()`	Recherche une expression régulière dans une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`replace()`	Remplace une expression régulière dans une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`search()`	Effectue une recherche dans une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`slice()`	Extrait une sous-chaîne de caractères, en fonction d'un caractère initial et d'un caractère final.	Internet Explorer, Mozilla, Firefox, Opéra.
`small()`	Diminue d'un niveau la taille de la police utilisée dans un document HTML.	Internet Explorer, Mozilla, Firefox, Opéra.
`split()`	Découpe une chaîne de caractères en fonction d'un séparateur passé en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`strike()`	Affiche une chaîne de caractères en mode barré.	Internet Explorer, Mozilla, Firefox, Opéra.
`sub()`	Affiche une chaîne de caractères en indice.	Internet Explorer, Mozilla, Firefox, Opéra.
`substr()`	Renvoie une sous-chaîne de caractères, en fonction d'une position et d'une longueur passées en argument.	Internet Explorer, Mozilla, Firefox, Opéra.

`substring()`	Renvoie une sous-chaîne de caractères, en fonction d'une position passée en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`sup()`	Affiche une chaîne de caractères en exposant.	Internet Explorer, Mozilla, Firefox, Opéra.
`toLowerCase()`	Convertit en minuscules une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`toString()`	Tente de convertir un objet en chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`toUpperCase()`	Convertit en majuscules une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher dans une page une chaîne de caractères selon différents modes d'affichage (gras, italique, clignotant, rouge, caractères barrés et de 7 tailles différentes) :




```
<script language="javascript">
var texte=prompt("Saisissez une chaîne de caractères :","Saisissez
ici votre chaîne de caractères");
document.write("Voici la chaîne de caractères en gras :<br>");
document.write(texte.bold());
document.write("Voici la chaîne de caractères en italique : <br>");
document.write(texte.italics());
document.write("Voici la chaîne de caractères en caractères
clignotants :" <br>);
document.write(texte.blink());
document.write("Voici la chaîne de caractères en rouge : <br>");
document.write(texte.fontcolor("red"));
document.write("Voici la chaîne de caractères barrée : <br>");
document.write(texte.strike());
document.write("Voici la chaîne de caractères en différentes tailles :");
for (i=0;i<7;i++) {
document.write(texte.fontsize(i));
}
</script>
```

Ici, la plupart des méthodes liées à HTML de l'objet String sont passées en revue. Ces méthodes doivent être utilisées avec parcimonie, car elles ne respectent pas le standard du W3C. Il vaut mieux leur préférer une mise en forme à l'aide des CSS (*Cascade Sheet Style* ou feuilles de style en cascade). Néanmoins, le script présente les fonctionnalités disponibles et il débute par une série d'affichage utilisant différentes propriétés de texte, puis se poursuit avec une boucle permettant d'afficher le texte avec une taille de plus en plus grande, à chaque passage de 1 à 7 correspondant aux tailles HTML définies par le W3C (*World Wide Web Consortium* : organisme de standardisation des technologies de l'Internet).

Exemple : afficher la longueur d'une chaîne de caractères saisie dans un champ et convertir en majuscules puis en minuscules le contenu de ce champ par un clic sur un bouton. La même chaîne de caractères sera ensuite convertie avec une majuscule pour débuter et le reste en minuscules.




```
<html>
<head>
<title>Conversion de champ en majuscules</title>
<script language="javascript">
function convertirmaj() {
longueur=document.form1.texte.value.length;
alert("La chaîne de caractères à convertir est longue de :
"+longueur+" caractères");docment.form1.majuscules.value=
document.form1.texte.value.toUpperCase();
}
function convertirmin() {
document.form1.minuscules.value=
document.form1.majuscules.value.toLowerCase();
}
function convert() {
document.form1.conversion.value=
document.form1.minuscules.value.charAt(0).toUpperCase()+
document.form1.minuscules.value.substring(1,longueur);
}
</script>
</head>
<body>
<form name="form1" method="post" action="">
  <p> </p>
  <table width="400" border="0">
    <tr>
      <td width="125"><input name="texte" type="text" id="texte"
value="Saisir un texte ici" size="20" maxlength="20"></td>
      <td width="265"> </td>
    </tr>
    <tr>
      <td><input name="majuscules" type="text" id="majuscules"
size="20" maxlength="20"></td>
      <td><input type="button" name="Submit" value="MAJUSCULES"
onClick="convertirmaj()"></td>
    </tr>
    <tr>
      <td><input name="minuscules" type="text" id="minuscules"
size="20" maxlength="20"></td>
      <td><input type="button" name="Submit2" value="Minuscules"
onClick="convertirmin()"></td>
    </tr>
    <tr>
      <td><input name="conversion" type="text" id="conversion"
size="20" maxlength="20"></td>
      <td><input type="button" name="Submit3" value="1ere Lettre"
onClick="convert()"></td>
    </tr>
  </table>
</form>
```

```
</body>
</html>
```

Le script se divise en plusieurs fonctions. La première, (convertirmaj()), permet de calculer la longueur de la chaîne saisie et de l'afficher, dans un autre champ du formulaire, tout en majuscules. Le calcul de la longueur de la chaîne s'effectue facilement, grâce à la propriété length appliquée à la valeur du champ de formulaire. Cette longueur est, ensuite, affichée dans une boîte de dialogue. La méthode toUpperCase() permet ensuite la conversion vers une chaîne en majuscules. La seconde fonction, (convertirmin()), utilise toLowerCase pour transformer, à l'inverse, la chaîne en minuscules. La troisième fonction, (convertir()), utilise le champ minuscules (c'est-à-dire celui qui correspond obligatoirement à un champ composé de minuscules), et transforme son premier caractère en majuscule pour ajouter le reste du champ en minuscules (c'est-à-dire du second caractère identifié par 1 jusqu'à la fin de la chaîne calculée précédemment).

Exemple : extraire une chaîne de caractères d'une autre chaîne saisie par l'utilisateur en fonction d'un début et d'une fin d'extraction. Le résultat est affiché dans la page et avec des caractères de couleur rouge :










```
<script language="javascript">
var texte=prompt("Quel est le texte de départ ?", "Saisissez votre
texte ici");
var pos1=prompt("A partir de quelle position ?", " Saisissez le numéro
du caractère de départ");
var pos2=prompt("Jusqu'à quelle position ?", "Saisissez le numéro du
caractère de fin");
var extrait=texte.substring(pos1,pos2);
document.write("Voici la chaine extraite :
"+extrait.fontcolor("red"));
</script>
```

Le script affecte trois variables pour débuter :

- La variable texte correspond au texte saisi par l'utilisateur.
- La variable pos1 correspond à la position du premier caractère de la sous-chaîne à extraire.
- La variable pos2 correspond à la position du dernier caractère de la sous-chaîne à extraire.

Ensuite, il suffit d'appliquer la méthode substring() à la chaîne de caractères pour extraire une sous-chaîne à partir de la position du premier et du dernier caractère. La chaîne est, ensuite, convertie en rouge et affichée dans le document.

Exemple : effectuer un test sur une extension de fichier destiné à être transféré. Si l'extension n'est pas gif ou jpg, un message d'avertissement s'affiche.




```
<html>
<head>
<title>Verification de type de fichier upload</title>
<script language="javascript">
function controler() {
var name=document.form1.fileup.value;
var longueur=name.length;
ext=longueur-3;
extension=name.substr(ext,3)
if(extension=="gif"||extension=="jpg") {
alert("OK");
}
else {
alert("Désolé ! Ce type de fichier ne peut pas être transféré");
}
}
</script>
</head>
<body>
<form name="form1" id="form1">
Fichier à envoyer:
<input type="file" name="fileup">
<input type="button" value="Envoyer"
onClick="controler ()">
<br>
</form>
</body>
</html>
```

Ici, le script s'exécute, l'utilisateur clique sur le bouton **Envoyer**, après avoir choisi le fichier à partir du bouton **Parcourir**. À ce moment, le script récupère le nom du fichier sélectionné et le stocke dans une variable appelée name. Ensuite, il calcule la longueur de la chaîne de caractères correspondant au chemin du fichier. Étant donné que l'extension comporte trois caractères, vous en déduisez que la chaîne à extraire par la méthode substr() débute au nombre de caractères moins 3. La chaîne de caractères extraite est, ensuite, testée pour savoir si elle correspond à l'une ou l'autre des extensions autorisées. À ce moment, il est possible d'afficher l'un ou l'autre des messages dans une boîte de dialogue.

Exemple : tester la valeur d'un champ de formulaire pour savoir s'il contient une arobase et un point.




```
<html>
<head>
<title>Test de la présence d'une arobase</title>
<script language="javascript">
function mail() {
if (document.form1.adresse.value.indexOf("@")<0 ||
document.form1.adresse.value.indexOf(".")<0) {
alert("adresse mail invalide, vérifier la présence de l'arobase
et du point");
}
else {
alert("adresse mail valide, elle contient une arobase et un point");
}
}
</script>
</head>
<body>
<form name="form1" method="post" action="">
  <p> </p>
  <table width="400" border="0">
    <tr>
      <td width="125"><input name="adresse" type="text" id="adresse"
value="Saisir une adresse mail ici" size="30" maxlength="30"></td>
      <td width="265"><input type="button" name="Submit"
value="Test" onclick="mail()"></td>
    </tr>
    <tr>
      <td> </td>
      <td> </td>
    </tr>
  </table>
</form>
</body>
</html>
```

Ce script peut être utilisé pour contrôler dans un formulaire d'inscription si le champ e-mail a bien été complété. Malgré tout, il ne permet pas une vérification totale puisque la présence d'une arobase et d'un point ne suffit pas à valider d'une façon certaine une adresse e-mail. Ce script utilise la méthode IndexOf() de l'objet string qui permet de connaître la position d'un caractère dans une chaîne. Sa syntaxe est la suivante :

```
chaînedecaractères.indexOf(recherche,début) ;
```

Où chaînedecaractères représente la chaîne dans laquelle effectuer la recherche, recherche correspond à la chaîne à rechercher et début à la position du caractère à partir duquel il faut commencer la recherche. L'argument début indique le caractère de début de la recherche. S'il est omis, la recherche commence au début de la chaîne.

Si cette position est inférieure à zéro (étant donnée que la position du premier caractère correspond à zéro), c'est que le caractère n'est pas présent dans le champ. Un message d'avertissement est alors renvoyé. D'autres méthodes peuvent être employées pour vérifier ce type de champ comme les expressions régulières, qui sont expliquées un peu plus loin.

6. Date

L'objet Date est un objet inclus de façon native dans JavaScript, c'est-à-dire que vous pouvez l'appeler à tout moment. Il permet de gérer le calcul du temps jusqu'à une précision de la milliseconde. Les dates sont représentées par un nombre compris entre -100 000 000 et +100 000 000 avec comme référence le 1er janvier 1970. L'objet Date n'a pas de propriété. Pour manipuler l'objet Date, il faut en créer une instance dans une variable. Par contre, l'objet Date possède trois types de méthode pour lire, écrire ou convertir. Il existe en plus un système de minuterie permettant de gérer les intervalles de temps. À noter toutefois, que le terme d'écriture ne correspond pas au fait de changer l'heure système, qui est tout à fait impossible.

a. Les méthodes de lecture

Méthode	Résultat	Reconnu par
`getDate()`	Renvoie le numéro du jour du mois entre 1 et 31.	Internet Explorer, Mozilla, Firefox, Opéra.
`getDay()`	Renvoie le numéro du jour de la semaine sachant que zéro correspond au dimanche et six au samedi.	Internet Explorer, Mozilla, Firefox, Opéra.
`getFullYear()`	Renvoie un nombre de quatre chiffres correspondant à l'année en cours.	Internet Explorer, Mozilla, Firefox, Opéra.
`getHours()`	Renvoie l'heure courante sachant que zéro correspond à minuit.	Internet Explorer, Mozilla, Firefox, Opéra.
`getMilliseconds()`	Renvoie un nombre en millisecondes correspondant à l'heure locale.	Internet Explorer, Mozilla, Firefox, Opéra.
`getMinutes()`	Renvoie le nombre de minutes de l'heure courante.	Internet Explorer, Mozilla, Firefox, Opéra.
`getMonth()`	Renvoie le numéro du mois courant sachant que zéro correspond à janvier et onze à décembre.	Internet Explorer, Mozilla, Firefox, Opéra.
`getSeconds()`	Renvoie le nombre de secondes de l'heure courante.	Internet Explorer, Mozilla, Firefox, Opéra.
`getTime()`	Renvoie le nombre de millisecondes écoulées depuis le 1er janvier 1970.	Internet Explorer, Mozilla, Firefox, Opéra.
`getTimezoneOffset()`	Renvoie le nombre de minutes séparant le lieu de l'exécution du script, du méridien de Greenwich.	Internet Explorer, Mozilla, Firefox, Opéra.
`getUTCDate()`	Renvoie le nombre du jour du mois exprimé en coordonnées UTC (*Temps Universel*	Internet Explorer, Mozilla, Firefox, Opéra.

	Coordonné).	
`getUTCDay()`	Renvoie le numéro du jour de la semaine exprimé en coordonnées UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
`getUTCFullYear()`	Renvoie un nombre de quatre chiffres correspondant à l'année en cours, en coordonnées UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
`getUTCHours()`	Renvoie l'heure courante exprimée en coordonnées UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
`getUTCMilliseconds()`	Renvoie le nombre de millisecondes de l'heure courante exprimé en coordonnées UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
`getUTCMonth()`	Renvoie le numéro du mois courant exprimé en coordonnées UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
`getYear()`	Renvoie un nombre de deux chiffres pour les années antérieures à l'an deux mille et de quatre pour les années postérieures (exemple 99 pour 1999 et 2008 pour 2008).	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher la date courante dans un document HTML.




```
<html>
<head>
<title> propriétés et méthodes de Date </title>
<script language="javascript">
aujourdhui=new Date;
jour=aujourdhui.getDate();
if (jour<10) {
jour=”0”+jour;
}
alert("numéro de mois : "+aujourdhui.getMonth());
mois=aujourdhui.getMonth()+1;
annee=aujourdhui.getFullYear();
document.write("Nous sommes le "+jour+"/"+mois+"/"+annee);
</script>
</head>
<body>
```

```
</body>
</html>
```

La première étape consiste à créer un objet Date aujourd'hui permettant de récupérer la date du jour. Ensuite, il est possible de récupérer le jour, le mois et l'année d'aujourd'hui. Une première particularité réside dans la gestion du numéro du jour. En effet pour améliorer l'affichage, il est préférable d'ajouter 0 devant les numéros inférieurs à 10. Une autre particularité se situe dans la gestion des numéros des mois, car étant donné que la numérotation des mois de l'année débute (comme toujours en JavaScript) par zéro, il convient d'ajouter un à la variable mois pour l'affichage dans le document.

Les mois s'affichent en nombre. Aussi, si vous souhaitez les obtenir en lettres, vous pouvez utiliser une structure de contrôle switch pour les déterminer en fonction de la variable mois. Le même genre d'instruction peut être utilisé pour déterminer le jour de la semaine avec la méthode getDay().




```
<html>
<head>
<title>propriétés et méthodes de Date</title>
<script language="javascript">
aujourdhui=new Date;
alert(aujourdhui);
nombrejour=aujourdhui.getDay();
if (nombrejour<10) {
nombrejour="0"+nombrejour ;
}
alert("Le nombre du jour est : "+nombrejour);
jour=aujourdhui.getDate();
alert("numéro de mois : "+aujourdhui.getMonth());
mois=aujourdhui.getMonth()+1;
alert("numéro de mois après ajout : "+mois);
annee=aujourdhui.getFullYear();
alert(annee);
switch (nombrejour) {
case 1 :
nomjour="lundi";
break;
case 2 :
nomjour="mardi";
break;
case 3 :
nomjour="mercredi";
break;
case 4 :
nomjour="jeudi";
break;
case 5 :
nomjour="vendredi";
break;
case 6 :
nomjour="samedi";
break;
case 7 :
nomjour="dimanche";
break;
}
alert(nomjour);
switch (mois) {
```

```
case 1 :
nomois="janvier";
break;
case 2 :
nomois="février";
break;
case 3 :
nomois="mars";
break;
case 4 :
nomois="avril";
break;
case 5 :
nomois="mai";
break;
case 6 :
nomois="juin";
break;
case 7 :
nomois="juillet";
break;
case 8 :
nomois="août";
break;
case 9 :
nomois="septembre";
break;
case 10 :
nomois="octobre";
break;
case 11 :
nomois="novembre";
break;
case 12 :
nomois="décembre";
break;
}
document.write("Nous sommes le "+nomjour+" "+jour+" "+nomois+"
"+annee);
</script>
</head>
<body>
</body>
</html>
```

Ce script, bien qu'un peu lourd, permet d'afficher les mois de manière littérale en améliorant ainsi la lisibilité du résultat.

b. Les méthodes d'écriture

Méthode	Résultat	Reconnu par
`setDate()`	Fixe la valeur du jour du mois.	Internet Explorer, Mozilla, Firefox, Opéra.
`setFullYear()`	Fixe la valeur de l'année.	Internet Explorer, Mozilla, Firefox, Opéra.
`setHours()`	Fixe la valeur de l'heure.	Internet Explorer, Mozilla, Firefox, Opéra.
`setMilliseconds()`	Fixe la valeur de la milliseconde.	Internet Explorer, Mozilla, Firefox, Opéra.

setMinutes()	Fixe la valeur des minutes.	Internet Explorer, Mozilla, Firefox, Opéra.
setMonth()	Fixe la valeur du numéro du mois.	Internet Explorer, Mozilla, Firefox, Opéra.
setSeconds()	Fixe la valeur des secondes.	Internet Explorer, Mozilla, Firefox, Opéra.
setTime()	Fixe la date.	Internet Explorer, Mozilla, Firefox, Opéra.
setUTCDate()	Fixe la date selon UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
setUTCFullYear()	Fixe la valeur de l'année selon UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
setUTCHours()	Fixe la valeur entre 0 et 23 de l'heure selon UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
setUTCMilliseconds()	Fixe la valeur entre 0 et 999 des millisecondes selon UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
setUTCMinutes()	Fixe la valeur entre 0 et 59 des minutes selon UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
setUTCMonth()	Fixe la valeur entre 0 et 11 du mois selon UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
setUTCSeconds()	Fixe la valeur entre 0 et 59 des secondes selon UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
setYear()	Fixe la valeur de l'année.	Internet Explorer, Mozilla, Firefox, Opéra.

c. Les méthodes de conversion

Méthode	Résultat	Reconnu par
parse()	Convertit un objet Date en nombre de millisecondes depuis le 1er janvier 1970.	Internet Explorer, Mozilla, Firefox, Opéra.
toLocaleString()	Convertit l'objet Date en une chaîne de caractères selon le format local.	Mozilla, Firefox, Opéra.

toString()	Convertit l'objet Date en une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
toUTCString()	Convertit l'objet Date en une chaîne de caractères selon le format UTC.	Internet Explorer, Mozilla, Firefox, Opéra.
UTC()	Convertit en millisecondes la différence entre une date et le 1er janvier 1970.	Internet Explorer, Mozilla, Firefox, Opéra.

d. Les minuteries

Il existe quatre méthodes particulières qui permettent de gérer une minuterie :

Méthode	Résultat	Reconnu par
setInterval()	Déclenche l'exécution d'une série d'instructions en fonction d'une fréquence en millisecondes passée en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
clearInterval()	Stoppe la minuterie lancée par setInterval().	Internet Explorer, Mozilla, Firefox, Opéra.
setTimeout()	Retarde le déclenchement d'une série d'instructions en fonction d'une durée en millisecondes passée en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
clearTimeout()	Stoppe la minuterie lancée par setTimeout().	Internet Explorer, Mozilla, Firefox, Opéra.

La syntaxe de `setInterval()` est la suivante :

```
setInterval(action, période, arguments éventuels de l'action) ;
```

SetInterval est parfois mal reconnu, notamment par les navigateurs dont la version est un peu ancienne. Il est préférable d'utiliser SetTimeout() lorsque vous en avez le choix.

La syntaxe de `clearInterval()` est la suivante :

```
clearInterval(nom de la minuterie) ;
```

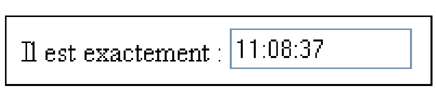
La syntaxe de `setTimeout()` est le suivante :

```
setTimeout(action,delai, arguments éventuels de l'action) ;
```

La syntaxe de `clearTimeout()` est la suivante :

```
clearTimeout(nom de la minuterie) ;
```

Exemple : créer une horloge simple affichant les heures, les minutes et les secondes dans un champ de formulaire :

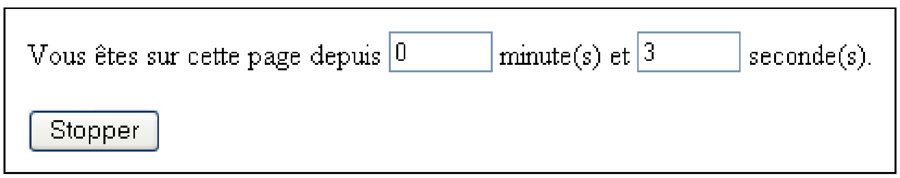



```
<html>
<head>
<head>
<title>Horloge</title>
<script language="JavaScript">
```

```
function Horloge() {
maintenant=new Date();
heures=maintenant.getHours();
minutes=maintenant.getMinutes();
secondes=maintenant.getSeconds();
if(heures<10) {
heures="0"+heures;
}
if(minutes<10) {
minutes="0"+minutes;
}
if(secondes<10) {
secondes="0"+secondes;
}
document.form1.Pendule.value=heures+":"+minutes+":"+secondes;
setTimeout("Horloge()", 1000);
}
</script>
</head>
<body onLoad="Horloge()">
<form name="form1" method="post" action="">
Il est exactement :
<input name="Pendule" type="text" id="Pendule" size="12">
</form>
</body>
</html>
```

Pour réussir à manipuler l'objet Date, il faut créer une nouvelle instance de cet objet. Puis, il est possible d'en extraire les heures, les minutes et les secondes. Ensuite, il faut traiter l'affichage pour les valeurs inférieures à 10 et leur ajouter un zéro devant si nécessaire, pour obtenir 01 à la place de 1, par exemple. Ensuite, il est possible d'afficher le contenu du champ de formulaire et de demander le rechargement de la fonction Horloge au bout de 1000 millisecondes (soit une seconde). Ainsi, l'horloge se modifiera toutes les secondes.

Exemple : afficher la durée de la visite d'une page en secondes, lors d'un clic sur un bouton permettant l'arrêt du compteur.

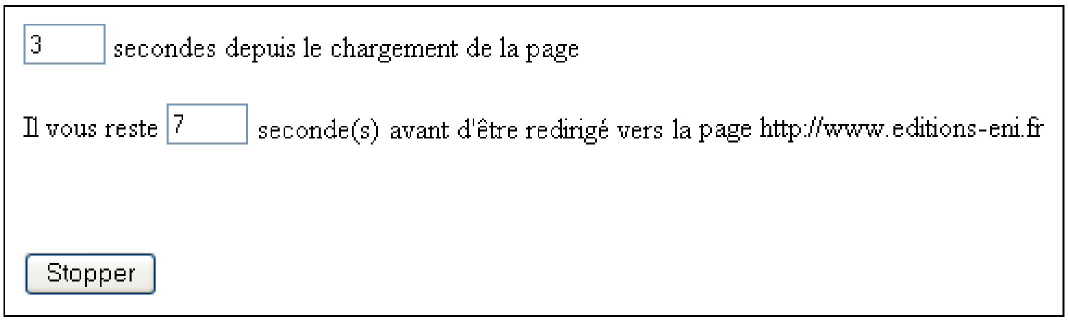



```
<html>
<head>
<title>Les minuteries</title>
<script language="javascript">
var minute(s)=0,seconde(s)=0;
function chrono() {
document.form1.minute(s).value=minute(s);
document.form1.seconde(s).value=seconde(s);
seconde(s)++;
if(seconde(s)==60) {
minute(s)=minute(s)+1;
seconde(s)=0;
}
compteur=setTimeout('chrono()',1000);
}
function stopper() {
clearInterval(seconde(s));
}
</script>
</head>
<body onLoad="chrono()">
<form name="form1" method="post" action="">
  Vous &ecirc;tes sur cette page depuis
    <input name="minute(s)" type="text" id="minute(s)" size="5"
maxlength="5">
    minute(s) et
    <input name="seconde(s)" type="text" id="seconde(s)" size="5"
```

```
maxlength="5">
    seconde(s).
</form>
<p>
  <input type="button" name="Submit" value="Stopper"
onclick="stopper()">
</p>
<p> </p>
</body>
</html>
```

Ce script s’exécute au chargement de la page. Les variables, seconde(s) et minute(s), sont initialisées, dès le début, afin de toujours partir avec des valeurs égales à 0. Ensuite, il est nécessaire d’écrire leur valeur dans les champs correspondants, avant de débuter l’incrémentation des compteurs. La variable nommée « seconde(s) » est incrémentée puis testée pour savoir si elle est égale à 60. Dans ce cas, il faut incrémenter la variable, nommée minute(s), puisque cela représente une nouvelle minute. Par la même occasion, il convient de réinitialiser les secondes à 0. La minuterie compteur permet, ensuite, de recommencer le traitement toutes les secondes. Le bouton **Stopper** lance l’exécution de la fonction du même nom qui permet d’interrompre la minuterie.

Exemple : réorienter un utilisateur sur la page http://www.editions-eni.fr au bout de dix secondes de visite sur une page. L’utilisateur dispose d’un compteur totalisant le nombre de secondes de la visite et d’un compteur à rebours égrenant les dernières secondes avant la redirection. Enfin, un bouton permet de stopper le compteur.




```
<html>
<head>
<title>Les minuteries</title>
<script language="javascript">
var i=0;
function chrono() {
i++;
document.form1.chrono.value=i;
document.form1.rebours.value=10-i;
if (i==10) {
window.location="http://www.editions-eni.fr";
}
else {
}
compteur=setTimeout(’chrono()’,1000);
}
function stopper() {
clearTimeout(compteur);
}
</script>
</head>
<body>
<body onLoad="chrono()">
<form name="form1" method="post" action="">
  <label for="textfield"></label>
  <p><input name="chrono" type="text" id="chrono" size="3"
maxlength="3">
  seconde(s) depuis le chargement de la page</p>
  <p>
    <label for="textfield"></label>
    Il vous reste
    <input name="rebours" type="text" id="rebours" size="3">
  seconde(s) avant d’&ecirc;tre redirig&eacute; vers la page
http://www.editions-eni.fr</p>
  <p> </p>
```

```
  <p>
    <label for="Submit"></label>
    <input type="button" name="Submit" value="Stopper" id="Submit"
onClick="stopper()">
  </p>
</form>
</body>
</html>
```

Ce script s'exécute au chargement de la page. La fonction chrono va permettre d'incrémenter une variable (i) et donc, simultanément, de décrémenter un compteur à rebours (document.form1.rebours.value=10-i). Le test sur (i) permet de savoir si les dix secondes ont été atteintes sachant que la fonction chrono se renouvelle toutes les secondes grâce à l'emploi de SetTimeOut. Si le test est vérifié, la redirection s'effectue immédiatement après, grâce à la propriété location de l'objet window et correspond à l'URL passée en argument. La fonction stopper() s'exécute après un clic sur le bouton **Stopper** et permet d'arrêter le compteur.

7. L'objet Array

Lorsqu'il s'agit de manipuler une grande quantité d'informations, le recours aux seules variables n'est parfois pas suffisant. L'usage de tableaux afin de stocker ces informations est, alors, d'un grand secours. Les tableaux JavaScript n'ont rien à voir avec les tableaux HTML, leur rôle n'est pas de présenter l'information mais de la stocker et de la mettre à disposition pour la manipuler (ajouter, supprimer, trier, etc.). Un tableau ne peut pas recevoir plus de 256 valeurs, mais les valeurs stockées dans celui-ci peuvent être de type différent (texte, numérique, etc.). L'objet Array représente donc à lui seul, une variable à l'intérieur de laquelle sont stockées des informations identifiées par un numéro d'indice débutant par zéro (comme bien souvent en JavaScript).

L'objet Array est un objet natif (c'est-à-dire appartenant au core) de JavaScript, qui dispose de dix méthodes. Pour utiliser un tableau, il convient tout d'abord de le déclarer. Il existe plusieurs syntaxes pour la déclaration d'un tableau.

Il est, par exemple, possible de déclarer un tableau et le nombre d'éléments qu'il contient par l'instruction suivante :

```
var tableau=new Array(5) ;
```

Mais l'indication du nombre d'éléments du tableau n'est pas obligatoire, en effet, JavaScript gère les tableaux de manière dynamique et s'adapte donc, au nombre d'éléments qui lui est fourni. Ainsi, la syntaxe suivante convient également :

```
var tableau =new Array() ;
```

Attention à ne pas oublier les parenthèses ouvrantes et fermantes après Array.

Puis, vous affectez les valeurs dans le tableau par l'instruction suivante :

```
var tableau= ["valeur1", "valeur2", "valeur3", "valeur4"] ;
```

Tout comme avec les variables, il est possible de réaliser la déclaration et l'affectation simultanément. Donc, la syntaxe suivante est aussi valide :

```
var tableau=new Array["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"] ;
```

L'accès aux valeurs stockées dans le tableau se fait par leur numéro d'indice :

Ainsi, `document.write(tableau[4])` ; affichera vendredi

Évidemment, le numéro d'indice peut provenir d'un compteur d'une boucle, ce qui permet de passer en revue l'ensemble des éléments du tableau.

Exemple : afficher dans un document HTML les jours de la semaine, passés en valeur d'un tableau.

lundi mardi mercredi jeudi vendredi samedi dimanche

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"] ;
```

```
for (i=0;i<7;i++) {
document.write(semainier[i]+" ");
}
</script>
```

Le script utilise un compteur et une boucle pour afficher le contenu d'un tableau dont vous connaissez le nombre d'éléments.

Dans le cas où vous ne connaissez pas le nombre d'éléments d'un tableau, il est possible d'utiliser la propriété length, correspondant au nombre de valeurs du tableau. Le script devient alors :

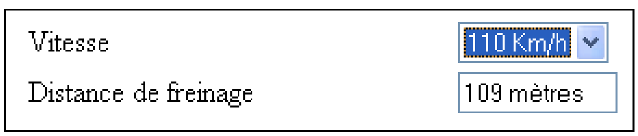
```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"] ;
var longueur=semainier.length;
for (i=0;i<longueur;i++) {
document.write(semainier[i]+" ");
}
</script>
```

Une autre possibilité d'affectation consiste à utiliser une autre forme de l'objet Array, les tableaux associatifs, qui permettent d'associer une valeur à une autre. Le principe consiste à rattacher un indice du tableau à une valeur. Le rattachement s'effectue, après la déclaration du tableau, par la syntaxe suivante :

```
Nomtableau[valeur1]= valeur2;
```

Exemple : afficher dans un champ de formulaire une distance de freinage, en fonction d'une vitesse choisie dans une liste déroulante et selon la matrice suivante :

50 Km/h	110 Km/h	130 Km/h
23 mètres	109 mètres	152 mètres




```
<html>
<head>
<title>Tableaux associatifs</title>
<script language="javascript">
function reponse() {
var distance=new Array();
distance[0]="23 mètres";
distance[1]="109 mètres";
distance[2]="152 mètres";
vitesse=document.form1.vitesse.selectedIndex;
document.form1.distance.value=distance[vitesse];
}
</script>
</head>
<body>
<form name="form1" method="post" action="">
  <table width="400" border="0">
    <tr>
      <td>Vitesse</td>
      <td><select name="vitesse" id="vitesse" onChange="reponse()">
          <option value="50">50 Km/h</option>
          <option value="110">110 Km/h</option>
          <option value="130">130 Km/h</option>
        </select>      </td>
    </tr>
    <tr>
      <td>Distance de freinage </td>
      <td><input name="distance" type="text" id="distance" size="10"
maxlength="10"></td>
```

```
    </tr>
  </table>
</form>
</body>
</html>
```

Les champs de formulaire sont initialisés au chargement de la page.

Le script s'exécute au moment du changement de valeur dans la liste déroulante. Il débute par l'élaboration du tableau associatif comprenant trois éléments. Ensuite, il faut déterminer le choix effectué par l'utilisateur en stockant dans une variable (ici vitesse), le numéro d'indice de ce choix. En fonction de ce numéro, il est facile de retourner la valeur correspondante dans le champ distance du formulaire.

a. Les propriétés

Des trois propriétés, seul length est très souvent rencontré dans les scripts. Il permet ainsi de déterminer le nombre d'itérations d'une boucle, correspondant au nombre d'éléments du tableau.

Propriété	Résultat	Reconnu par
constructor	Indique comment a été créé un objet référencé.	Internet Explorer, Mozilla, Firefox, Opéra.
length	Correspond au nombre d'éléments du tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
prototype	Permet d'ajouter des propriétés personnalisées à l'objet.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher le nombre d'éléments d'un tableau comportant les jours de la semaine (semainier).

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
longueur=semainier.length;
alert("La semaine est composée de "+longueur+" jours");
</script>
```

Après la définition des éléments du tableau, le calcul du nombre d'éléments le composant est facile par la propriété length.

b. Les méthodes

Les méthodes applicables à l'objet Array sont très utiles et disposent d'une syntaxe relativement simple à retenir :

```
nomdutableau.méthode(arguments) ;
```

Méthode	Résultat	Reconnu par
concat()	Concatène deux tableaux en un.	Internet Explorer, Mozilla, Firefox, Opéra.
join()	Crée une chaîne de caractères à partir des éléments du tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
pop()	Supprime le dernier élément d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
push()	Ajoute des éléments en fin de tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
reverse()	Inverse l'ordre des éléments d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.

`shift()`	Supprime le premier élément d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
`slice()`	Retourne ou ajoute une partie d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
`sort()`	Trie les éléments d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
`splice()`	Ajoute, supprime ou remplace les éléments d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
`toString()`	Convertit un tableau en une chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`unshift()`	Ajoute un élément au début d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.
`valueOf()`	Retourne un élément précis d'un tableau.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher dans un document HTML, le contenu d'un tableau composé des jours de la semaine suivi chacun de la chaîne de caractères « puis ».

lundi puis mardi puis mercredi puis jeudi puis vendredi puis samedi puis dimanche

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"] ;
tableau=semainier.join(" puis ");
document.write(tableau);
</script>
```

L'utilisation de la méthode join() permet d'adjoindre une chaîne de caractères (ici, le mot puis), entre les éléments du tableau. Mais, cet argument est facultatif. La chaîne de caractères correspond, alors, aux seuls éléments du tableau séparés par des virgules et il est possible de la traiter comme toute autre chaîne de caractères.

Exemple : afficher en rouge les éléments d'un tableau après les avoir converti en chaîne de caractères.

Voici la semaine en débutant par lundi : lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche
lundi mardi mercredi jeudi vendredi samedi dimanche

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"] ;
tableau=semainier.join(" ");
document.write(tableau.fontcolor("red"));
</script>
```

Exemple : afficher le contenu d'une concaténation de deux tableaux, le premier correspondant aux premiers jours de la semaine, le second aux derniers.

lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche

```
<script language="javascript">
var debut,fin=new Array();
var debut=["lundi", "mardi", "mercredi"] ;
var fin=["jeudi", "vendredi", "samedi", "dimanche"];
var semainier=debut.concat(fin);
document.write(semainier);
```

```
</script>
```

Les deux tableaux, comprenant des valeurs différentes, sont facilement concaténés à l’aide de cette méthode.

Les méthodes qui suivent se fondent sur le principe de la structure de pile. C’est-à-dire que, pour ajouter ou supprimer des éléments dans un tableau, vous ne pouvez le faire que par le haut ou par le bas. Autrement dit, les éléments peuvent être extraits soit dans l’ordre dans lequel ils ont été placés, soit dans l’ordre inverse.

Exemple : afficher, dans la page HTML, l’ensemble des jours de la semaine puis sans le dimanche et enfin sans le samedi.

Voici la semaine complète : lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche
Voici la semaine sans dimanche : lundi,mardi,mercredi,jeudi,vendredi,samedi
Puis sans samedi : lundi,mardi,mercredi,jeudi,vendredi

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
document.write("Voici la semaine complète : "+semainier+"<br>");
semainier.pop();
document.write("Voici la semaine sans dimanche : "+semainier+"<br>");
semainier.pop();
document.write("Puis sans samedi : "+semainier+"<br>");
</script>
```

Il faut faire usage ici, de la méthode pop() dont la syntaxe d’utilisation est la suivante :

```
nomdutableau.pop() ;
```

L’usage successif de la méthode pop() supprime les éléments du tableau les uns après les autres, en partant du dernier élément.

Exemple : afficher, dans la page HTML, les jours de la semaine dans l’ordre puis dans l’ordre inverse.

Voici les jours de la semaine dans l'ordre : lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche
Voici les jours de la semaine dans le désordre : dimanche,samedi,vendredi,jeudi,mercredi,mardi,lundi

```
<script language="javascript">
var semainier=new Array();
var endroit=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
document.write("Voici les jours de la semaine dans l’ordre :
"+endroit+"<br>");
var envers=endroit.reverse();
document.write("Voici les jours de la semaine dans le désordre :
"+envers+"<br>");
</script>
```

La méthode reverse() inverse l’ordonnancement des valeurs d’un tableau.

Exemple : afficher, dans une page HTML, l’ensemble des jours de la semaine en partant du lundi, puis en partant du dimanche.

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
document.write("Voici la semaine en débutant par lundi "+semainier+"<br>");
semainier.reverse();
document.write("Voici la semaine inversée : "+semainier+"<br>");
</script>
```

Exemple : afficher, dans une page HTML, l’ensemble des jours de la semaine puis sans le lundi et enfin sans le mardi.

Voici la semaine complète : lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche
Voici la semaine sans lundi : mardi,mercredi,jeudi,vendredi,samedi,dimanche
Puis sans mardi : mercredi,jeudi,vendredi,samedi,dimanche

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
document.write("Voici la semaine complète "+semainier+"<br>");
semainier.shift();
document.write("Voici la semaine sans lundi : "+semainier+"<br>");
semainier.shift();
document.write("Puis sans mardi : "+semainier+"<br>");
</script>
```

La méthode shift() fonctionne à l'identique de la méthode pop() mais dans l'autre sens (c'est-à-dire en supprimant le premier élément du tableau).

Exemple : afficher, dans une page HTML, uniquement les jours d'une semaine de travail (du lundi au vendredi).

La semaine de travail comprend les jours suivants : lundi,mardi,mercredi,jeudi,vendredi

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
travail=semainier.slice(0,5);
document.write("La semaine de travail comprend les jours suivant :
"+travail);
</script>
```

La méthode slice() utilise deux arguments, placés entre parenthèses. La syntaxe est donc la suivante :

```
nouveautableau=tableau.slice(n,m) ;
```

Où n correspond à l'indice inférieur des éléments à extraire et m à l'indice supérieur. Si m n'est pas renseigné, tous les éléments du tableau sont alors extraits et si m est égal à n, aucun des éléments du tableau n'est extrait.

Exemple : afficher, dans une page HTML, les jours de la semaine triés par ordre alphabétique.

Les jours de la semaine triés par ordre alphabétique : dimanche,jeudi,lundi,mardi,mercredi,samedi,vendredi
Les jours de la semaine triés en ordre inverse : vendredi,samedi,mercredi,mardi,lundi,jeudi,dimanche

```
<script language="javascript">
function compare(n,m) {
if(n>m) {
return -1;
}
else {
return 1;
}
}
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
croissant=semainier.sort();
document.write("Les jours de la semaine triés par ordre alphabétique :
"+croissant+"<br>");
decroissant=semainier.sort(compare);
document.write("Les jours de la semaine triés en ordre inverse :
"+decroissant+"<br>");
</script>
```

La méthode sort() dispose d'une fonction de comparaison permettant d'obtenir un ordre de tri croissant ou décroissant. La syntaxe d'utilisation est la suivante :

```
nouveautableau=tableau.sort(fonctiondecomparaison) ;
```

Si cette fonction n'est pas utilisée, c'est l'ordre alphabétique qui est appliqué. C'est le cas dans notre exemple, pour établir la variable nommée croissant. Par contre, pour utiliser l'ordre décroissant, il faut passer par une autre fonction renvoyant un résultat à partir d'un test. Ici, n n'étant pas supérieur à m, c'est la valeur -1 qui est retournée et qui permet de donner l'ordre de tri à la méthode sort().

La méthode sort() s'applique, par défaut, sur des éléments de type alphanumérique. Pour obtenir un tri d'éléments numériques, il faut passer une autre fonction comme fonction de comparaison.

Exemple : afficher, dans un document HTML, tout d'abord les deux premiers jours d'une semaine de travail (lundi et mardi), et sur une ligne suivante les deux derniers jours (jeudi et vendredi).

Les deux premiers jours de la semaine de travail : lundi,mardi
Les deux derniers jours de la semaine de travail : jeudi,vendredi

```
<script language="javascript">
var semainier=new Array();
var semainier=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
tranche1=semainier.splice(0,2);
tranche2=semainier.splice(1,2);
document.write("Les deux premiers jours de la semaine de travail :
"+tranche1+"<br>");
document.write("Les deux derniers jours de la semaine de travail :
"+tranche2);
</script>
```

La méthode splice() utilise plusieurs arguments afin d'extraire les éléments d'un tableau. Sa syntaxe est la suivante :

```
nouveautableau=tableau.splice(i,j) ;
```

Où i correspond à l'indice du premier élément concerné par l'application de la méthode et j correspond au nombre d'éléments à extraire.

La suppression des deux premiers éléments du tableau réaffecte les numéros indices (Mercredi reprend l'indice zéro après la première suppression (tranche 1)).

La méthode unshift() permet d'insérer un élément manquant en première position d'un tableau.

Exemple : afficher dans une boîte de dialogue les jours de la semaine sans le lundi, puis une autre avec l'ensemble des jours.

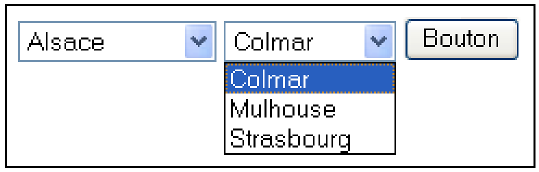
Voici la semaine sans le lundi : mardi,mercredi,jeudi,vendredi,samedi,dimanche
Voici la semaine complète : lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche

```
<script language="javascript">
var semainier=new Array();
var semainier=["mardi", "mercredi", "jeudi", "vendredi",
"samedi", "dimanche"];
document.write("Voici la semaine sans le lundi :
"+semainier+"<br>");
semainier.unshift("Lundi");
document.write("Voici la semaine complète : "+semainier);
</script>
```

Cette méthode ne dispose d'aucun argument et sa syntaxe est la suivante :

```
nouveautableau=tableau.unshift() ;
```

Exemple : créer une liste déroulante comprenant quelques villes de trois régions françaises à partir de la sélection d'une région dans une autre liste (Alsace : Strasbourg, Mulhouse, Colmar ; Ile-de-France : Paris, Evry, Cergy, Versailles ; Lorraine : Metz, Nancy, Verdun). Juste après le choix de la région, la liste des villes devra être proposée.




```
<html>
<head>
<title>Objet Array - Une liste à partir d'une autre</title>
<script language="javascript">
function listeville() {
var villeAlsace = new Array();
var villeIDF=new Array();
var villeLorraine=new Array();
villeAlsace=["Strasbourg","Mulhouse","Colmar"];
villeIDF=["Paris","Evry","Cergy","Versailles"];
villeLorraine=["Metz","Nancy","Verdun"];
if(document.form1.Region.value=="Alsace") {
villeAlsace.sort();
nbelements=villeAlsace.length;
for(i=0;i<nbelements;i++) {
document.form1.ville.options[i]=
new Option(villeAlsace[i],villeAlsace[i]); }
}
else if(document.form1.Region.value=="Ile-de-France") {
villeIDF.sort();
nbelements=villeIDF.length;
for(i=0;i<nbelements;i++) {
document.form1.ville.options[i]=new Option(villeIDF[i],villeIDF[i]); }
}
else {
villeLorraine.sort();
nbelements=villeLorraine.length;
for(i=0;i<nbelements;i++) {
document.form1.ville.options[i]=new Option(villeLorraine[i],
villeLorraine[i]);
}
}
}
</script>
</head>

<body onLoad="listeville()">
<form id="form1" name="form1" method="post" action="">
  <select name="Region" id="Region" onChange="listeville()">
    <option value="Alsace">Alsace</option>
    <option value="Ile-de-France">Ile-de-France</option>
    <option value="Lorraine">Lorraine</option>
  </select>
  <select name="ville" id="ville">
  </select>
  <input type="button" name="Submit" value="Bouton"
onClick="listeville()"/>
</form>
</body>
</html>
```

Le script s'exécute au moment d'un choix dans la liste des régions. C'est donc, l'évènement Onchange qui permet de lancer le script. La première étape est de créer et d'alimenter les trois tableaux correspondant aux villes des régions. Ensuite, vient le test permettant de déterminer la région choisie dans la première liste déroulante. Pour cela, il faut comparer la valeur choisie dans la liste avec chacun des noms des régions. Une fois la région choisie déterminée, un tri du tableau semble approprié pour présenter une liste triée des éléments. Étant donné que la liste pourra comporter d'autres valeurs par la suite, il est plus sûr de calculer le nombre d'éléments de la liste (variable nbelements). Il faut ensuite faire une boucle pour affecter les valeurs du tableau aux éléments composant la nouvelle liste. Cette boucle a pour limite le nombre total d'éléments à présenter. Il est alors possible, d'affecter de nouvelles valeurs aux éléments de la liste, grâce à l'instruction new Option.

c. Les tableaux multidimensionnels

Lorsque les données à stocker sont nombreuses, il est possible d'utiliser les tableaux à plusieurs entrées, aussi appelés multidimensionnels.

La construction de ce type de tableau nécessite la déclaration des deux tableaux imbriqués. Vous déclarez d'abord, un tableau de façon classique. Puis, vous réalisez l'affectation des valeurs en fonction de leur position (colonne, ligne) dans le tableau.

La syntaxe peut se résumer ainsi pour un tableau de 3 lignes et 3 colonnes :

```
var tableau=new Array(nombre de valeurs du tableau) ;
var tableau[0]=new Array(nombre de valeurs du tableau) ;
var tableau[1]=new Array(nombre de valeurs du tableau) ;
tableau[0][0]= "valeur1" ;
tableau[0][1]= "valeur2" ;
tableau[0][2]= "valeur3" ;
tableau[1][0]= "valeur1" ;
tableau[1][1]= "valeur2" ;
tableau[1][2]= "valeur3" ;
```

Exemple : afficher, dans une page HTML, les réflexions enregistrées du mercredi et du vendredi, toutes ces réflexions ayant été enregistrées préalablement, pour chaque jour de la semaine, dans un tableau à deux dimensions.

Voici le mot du mercredi : Jour des enfants
Voici le mot du vendredi : C'est le week-end

```
<script language="javascript">
var semainier=new Array(5);
semainier[0]=new Array(2);
semainier[1]=new Array(2);
semainier[2]=new Array(2);
semainier[3]=new Array(2);
semainier[4]=new Array(2);
semainier[0][0]="lundi";
semainier[0][1]="Au boulot";
semainier[1][0]="mardi";
semainier[1][1]="Ca va fort";
semainier[2][0]="mercredi";
semainier[2][1]="Jour des enfants";
semainier[3][0]="jeudi";
semainier[3][1]="Encore un effort";
semainier[4][0]="vendredi";
semainier[4][1]="C'est le week-end";
alert("Voici le mot du mercredi : "+semainier[2][1]);
alert("Voici le mot du vendredi : "+semainier[4][1]);
</script>
```

Exemple : afficher, dans une boîte de dialogue, la valeur stockée dans un tableau à deux dimensions.

	Janvier	**Février**	**Mars**
Produit A	10000	10500	12500
Produit B	10000	9500	8500
Produit C	11000	7500	8500




```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<title>Tableau multidimensionnel</title>
<script language="javascript">
function affiche() {
nomproduit=document.form1.Produit.value;
nommois=document.form1.mois.value;
var ca=new Array(3);ca[0]=new Array(2);ca[1]=new Array(2);
ca[2]=new Array(2);
ca[0][0]=10000;ca[0][1]=10500;ca[0][2]=12500;ca[1][0]=11000;ca[1][1]=
9500;ca[1][2]=8500;ca[2][0]=8000;ca[2][1]=7500;ca[2][2]=8500;
resultat=ca[nomproduit][nommois];
alert("Le chiffre d’affaires est de "+resultat+" euros.");
}
</script>
</head>
<body>

<form id="form1" name="form1" method="post" action="">
  <label for="select"></label>
  <p> </p>
  <label for="select"></label>
  <table width="400" border="0">
    <tr>
      <td><label for="select">Produit</label></td>
      <td><select name="Produit" id="Produit">
        <option value="0">Produit A</option>
        <option value="1">Produit B</option>
        <option value="2">Produit C</option>
      </select></td>
    </tr>
    <tr>
      <td>Mois</td>
      <td><label for="select"></label>
        <select name="mois" id="mois">
          <option value="0">janvier</option>
          <option value="1">F&eacute;vrier</option>
          <option value="2">mars</option>
        </select>      </td>
    </tr>
    <tr>
      <td colspan="2"><input type="button" name="Submit"
value="Afficher" onClick="affiche()" /></td>
    </tr>
  </table>
  <p>   </p>
</form>
</body>
</html>
```

La fonction affiche() s’exécute au moment du clic sur le bouton. Il faut tenir compte des valeurs sélectionnées dans les deux listes déroulantes, pour identifier les coordonnées de la valeur à afficher. Les deux variables nommois et nomproduit sont, donc, affectées, dès le début du script, avec les valeurs des listes déroulantes. Ensuite, il est

possible de déclarer les tableaux imbriqués et d'y affecter les valeurs. L'extraction de la valeur s'effectue en fonction des variables précédentes. L'affichage de la valeur extraite termine le script.

8. image

L'objet image se trouve sous l'objet document dans le modèle d'objet JavaScript. Il correspond à la balise HTML <img>. Cet objet est essentiel pour améliorer la présentation des pages web. L'objet image est souvent utilisé pour créer des roll-over (permutation d'image au passage de la souris) ou un album photo avec vignettes. Avec cet objet, vous pouvez accéder à toutes les images présentes dans la page HTML. L'objet image dispose de dix propriétés et d'une méthode.

a. Les propriétés

Propriété	Résultat	Reconnu par
`alt`	Correspond au texte à afficher dans le cas où le navigateur ne prendrait pas en charge les images ou encore, si la page est consultée par un public déficient visuel, équipé d'un appareillage ayant capacité à lire cette propriété. Le texte s'affiche dans une info-bulle lorsque la souris reste positionnée dessus durant quelques secondes.	Internet Explorer, Mozilla, Firefox, Opéra.
`border`	Indique la taille de la bordure figurant éventuellement autour de l'image.	Internet Explorer, Mozilla, Firefox, Opéra.
`complete`	Indique si l'image contenue dans la page est définitivement chargée. Renvoie true si c'est le cas.	Internet Explorer, Mozilla, Firefox, Opéra.
`fileSize`	Indique la taille de l'image en octets.	Mozilla, Firefox, Opéra.
`height`	Indique la hauteur de l'image en pixels.	Internet Explorer, Mozilla, Firefox, Opéra.
`hspace`	Correspond à l'espace en pixels entre une image et les éléments situés à droite et à gauche.	Internet Explorer, Mozilla, Firefox, Opéra.
`length`	Correspond au nombre d'images situées dans la page.	Internet Explorer, Mozilla, Firefox, Opéra.
`lowsrc`	Correspond au statut d'aperçu d'une image.	Internet Explorer, Mozilla, Firefox, Opéra.
`name`	Correspond au nom de l'image s'il existe.	Internet Explorer, Mozilla, Firefox, Opéra.
`src`	Correspond à l'adresse source (URL) du fichier de l'image. Cette propriété modifiable permet de nombreux effets, basés sur les images.	Internet Explorer, Mozilla, Firefox, Opéra.
	Correspond au titre de l'image. Tout comme la propriété alt, elle	

title	est intéressante pour les personnes disposant d'un appareillage de lecture, mais aussi pour le référencement des pages.	Internet Explorer, Mozilla, Firefox, Opéra.
vspace	Correspond à l'espace en pixels entre une image et les éléments situés en haut et en bas.	Internet Explorer, Mozilla, Firefox, Opéra.
width	Indique la largeur de l'image en pixels.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : Roll over avec deux images :



```
<head>
<title>Objet Image -rollover</title>
<script language="javascript">
Image1 = new Image(280,210);
Image1.src = "velo1.jpg";
Image2 = new Image(285,210);
Image2.src = "velo2.jpg";
function changeimage(numero,objet) {
document.images[numero].src = objet.src;
}
</script>
</head>
<body>
<img src="velo.jpg" width="285" height="210" onMouseOver=
"changeimage(0,Image2)" onMouseOut="changeimage(0,Image1)">
</body>
</html>
```

La première étape consiste à créer les objets images, qui seront nécessaires. L'objet image se construit avec l'instruction Image = new Image;.

Ici, l'image dispose de propriétés de largeur et de hauteur définies entre parenthèses et séparées par des virgules.

L'image à modifier correspond bien à l'indice 0, puisqu'en JavaScript (comme toujours) le premier numéro d'indice est 0 et non 1. À noter que la page ne comporte aucune autre image. C'est bien cette image qui prendra comme source l'objet image1 ou l'objet image2, en fonction de la position de la souris.

La seconde étape consiste à créer la fonction qui permet d'effectuer le changement d'image (ici function changeimage). Il suffit de faire correspondre la source à l'objet image, passé à la fonction dans l'instruction évènementielle `onMouseOver` et `onMouseOut`. Concrètement, lors du passage sur l'image (`onMouseOver`), la fonction changeimage est exécutée et reçoit comme paramètres l'indice 0 (correspondant à l'image à modifier), et l'objet (image1 ou image2). Ainsi, lors de l'instruction :

`document.images[numero].src = objet.src;` c'est l'image1 ou l'image2 qui est chargée.

Exemple : créer une bannière qui permute deux images, toutes les trois secondes.




```
<html>
<head>
<title>Images banniere</title>
<script language="javascript">
var limite=40;
var compteur=38;
function baniere() {
compteur++;
if (compteur>=limite) {
compteur=38;
}
document.images["velo"].src=compteur+".jpg";
return compteur;
}
function permute() {
 var id=setInterval("baniere()",3000);
}
</script>
</head>
<body onLoad="permute()">
<img src="velo1.jpg" name="velo" width="285" height="210">
</body>
</html>
```

Dans un premier temps, la fonction permute est exécutée au chargement de la page. Cette fonction de minuterie déclenche l'exécution de la fonction baniere, au bout de 3 secondes. Celle-ci incrémente, tout d'abord, une variable compteur, qui est ensuite testée pour savoir si elle est supérieure ou égale à la valeur de la variable globale limite (qui a pour valeur, dans notre exemple, 40). Si c'est le cas, la variable prend obligatoirement la valeur 38 grâce à l'instruction :

```
if (compteur>=limite) {
compteur=38;
}
```

L'image au format jpg ayant ce numéro apparaîtra alors. Pour terminer, l'instruction return permet de sortir de la fonction et de renvoyer la valeur de la variable compteur, pour une éventuelle incrémentation. La fonction permute recommence au bout de trois secondes et ainsi de suite.

9. history

L'objet history est un sous-objet de l'objet window. Il correspond à l'historique conservé dans le navigateur. Il dispose d'une propriété et de trois méthodes.

a. La propriété

Propriété	Résultat	Reconnu par
`length`	Correspond au nombre de pages visitées pour la fenêtre active.	Internet Explorer, Mozilla, Firefox, Opéra.

b. Les méthodes

Méthode	Résultat	Reconnu par
`back()`	Charge la dernière page visitée, présente dans l'historique.	Internet Explorer, Mozilla, Firefox, Opéra.
`forward()`	Charge la page suivante visitée, présente dans l'historique.	Internet Explorer, Mozilla, Firefox, Opéra.
`go()`	Se déplace dans l'historique des pages visitées, suivant un nombre de pages passé comme paramètre.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : créer trois pages permettant de naviguer entre elles et d'alimenter ainsi l'historique de navigation (ajouter des liens pour passer de la première à la seconde puis à la troisième). Sur la première, permettre l'affichage d'une boîte de dialogue contenant l'adresse de la page. Sur la seconde, ajouter deux boutons permettant d'avancer ou de reculer dans l'historique. Sur la dernière, ajouter un bouton permettant de connaître le nombre de liens mémorisés dans l'historique et un champ permettant de saisir un nombre de liens de retour en arrière.

Script de la première page :

```
<html>
<head>
<title>Page1</title>
<script language="javascript">
function adresse(){
adresse=location.href;
alert(adresse);
}
</script>
</head>

<body>
<div align="center">
  <p><strong>PAGE 1 </strong></p>
  <table width="100%" border="0">
    <tr>
      <td width="50%"><a href="page1.html"></a></td>
      <td width="50%"><div align="right"><a href="page2.html">Page 2
</a></div></td>
    </tr>
  </table>
  <p> </p>
  <p align="right"> </p>
  <p>
  <input type="submit" name="Submit" value="Adresse" onClick="adresse()">
</p>
</div>
</body>
</html>
```

Sur cette page, il n'y a qu'une seule fonction qui permet de récupérer l'URL de la page active. Il faut utiliser le lien

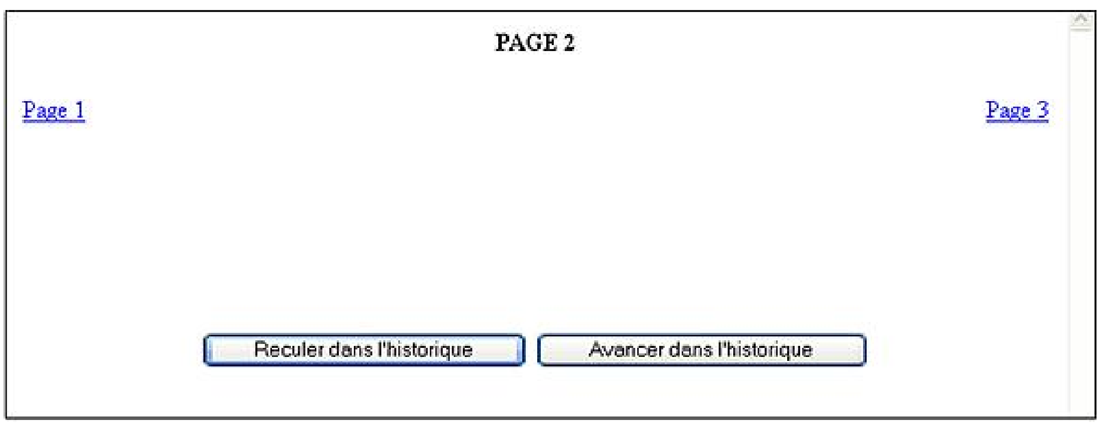
pour passer à la page suivante.

Script de la seconde page :

```
<html>
<head>
<title>Page2</title>
<script language="javascript">
function retour(){
window.history.back();
}
function avance() {
window.history.forward();
</script>
</head>

<body>
<div align="center">
  <p><strong>PAGE 2 </strong></p>
  <table width="100%" border="0">
    <tr>
      <td width="50%" height="29"><a href="page1.html">Page 1
</a></td>
      <td width="50%"><div align="right"><a href="page3.html">Page 3
</a></div></td>
    </tr>
  </table>
  <p> </p>
  <p align="center"> </p>
  <p> </p>
  <p>
  <input type="submit" name="Submit4" value="Reculer dans l'historique"
onClick="retour()">
  <input type="submit" name="Submit" value="Avancer dans l'historique"
onclick="avance()" />
  </p>
</div>
</body>
</html>
```

Sur cette seconde page, figurent les deux scripts permettant de naviguer dans l'historique en avançant ou en reculant.



Script de la troisième page :

```
<html>
<head>
<title>Page3</title>
<script language="javascript">
function compte(){
var nombre=window.history.length;
alert(nombre);
}
function reculer() {
```

```
var nbrecul="-"+document.form1.recul.value;
convertnbrecul=parseInt(nbrecul);
window.history.go(convertnbrecul);
}
</script>
</head>

<body>
<div align="center">
  <p><strong>PAGE 3 </strong></p>
  <p> </p>
  <form name="form1" method="post" action="">
    <table width="100%" border="0">
      <tr>
        <td width="50%"><a href="page2.html">Page 2 </a></td>
        <td width="50%"><div align="right">
            <input type="button" name="Submit" value="Reculer"
onClick="reculer()">
          de
          <input name="recul" type="text" id="recul" size="5"
maxlength="5">
          liens dans l'historique </div></td>
      </tr>
    </table>
    </form>
  <p align="right"> </p>
  <p>
  <input type="submit" name="Submit3" value="Compte Historique"
onClick="compte()">
  </p>
</div>
</body>
</html>
```

Cette troisième page permet d'afficher le nombre de liens mémorisés dans l'historique et d'effectuer un retour en arrière dans l'historique, suivant un nombre d'étapes saisi dans le champ recul. Étant donné qu'il s'agit d'un retour en arrière dans l'historique, le chiffre doit être précédé du signe moins. Pour cela, il faut créer une chaîne de caractères comprenant ce signe et la valeur du champ. Quant à la méthode go(), elle doit recevoir comme paramètre un chiffre. C'est la raison pour laquelle la méthode parseInt() est utilisée pour affecter la variable convertnbrecul.

> Pour tester ces différents scripts, il est préférable de vider l'historique du navigateur. Il est possible que l'exécution du script ne donne aucun résultat si le niveau de sécurité du navigateur est élevé.

10. location

Cet objet correspond à l'URL courante. Il est possible d'interroger l'URL actuellement chargée ou de la modifier, ce qui est nettement plus intéressant. S'il y a une modification, le navigateur charge la page en fonction de la nouvelle URL. L'objet location dispose de huit propriétés et de deux méthodes.

a. Les propriétés

Propriété	Résultat	Reconnu par
hash	Correspond à la chaîne de caractères représentant l'ancre nommée à l'intérieur d'une page.	Internet Explorer, Mozilla, Firefox, Opéra.
host	Correspond à la chaîne de caractères représentant la partie de l'URL, en partant de http jusqu'au numéro de port, mais sans ceux-ci.	Internet Explorer, Mozilla, Firefox, Opéra.
hostname	Correspond à la chaîne de caractères représentant la partie de l'URL, en partant	Internet Explorer, Mozilla, Firefox,

	de http jusqu'au numéro de port inclus.	Opéra.
`href`	Correspond à l'URL entière.	Internet Explorer, Mozilla, Firefox, Opéra.
`pathname`	Correspond à la partie suivant le nom d'hôte.	Internet Explorer, Mozilla, Firefox, Opéra.
`port`	Correspond au numéro éventuel de port de l'URL en cours (ex : http//www/serveur.com/page.html :8550).	Internet Explorer, Mozilla, Firefox, Opéra.
`protocol`	Correspond au protocole utilisé (ex : file pour les fichiers locaux et http pour ceux situés sur un serveur web).	Internet Explorer, Mozilla, Firefox, Opéra.
`search`	Correspond à la chaîne de recherche à partir du point d'interrogation, lors d'une requête par l'intermédiaire d'un bouton Submit.	Internet Explorer, Mozilla, Firefox, Opéra.

b. Les méthodes

Méthode	Résultat	Reconnu par
`reload()`	Correspond au bouton actualiser du navigateur, recharge la page actuelle.	Internet Explorer, Mozilla, Firefox, Opéra.
`replace()`	Remplace l'URL de la page actuelle par une autre ; de ce fait l'URL actuelle n'est pas stockée dans l'historique.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher le nom d'hôte du serveur, le chemin, le protocole utilisé ainsi que l'URL actuelle dans une boîte de dialogue. Ensuite, charger la page dont l'URL est : http://www.editions-eni.fr

Voici le nom d'hôte du serveur de la page actuelle : www.form-high-tech.com
Voici le chemin de la page actuelle : /Javascript/page.html
Voici le protocole utilisé pour la page actuelle : http:
Voici l'URL de la page actuelle : http://www.form-high-tech.com/Javascript/page.html

```
<html>
<head>
<title>Objet Location</title>
<script language="javascript">
document.write("Voici le nom d'hôte du serveur de la page actuelle :
"+window.location.hostname+"<br>");
document.write("Voici le chemin de la page actuelle :
"+window.location.pathname+"<br>");
document.write("Voici le protocole utilisé pour la page actuelle :
"+window.location.protocol+"<br>");
document.write("Voici l'URL de la page actuelle :
"+window.location.href+"<br>");
window.location.href="http://www.editions-eni.fr";
}
</script>
</head>
<body onLoad="changeURL()">
</body>
</html>
```

11. link

Cet objet correspond aux liens éventuels définis dans un document HTML. Il est possible de compter, lire, modifier les liens présents dans la page. Cet objet dispose de neuf propriétés.

Les propriétés

Propriété	Résultat	Reconnu par
`name`	Correspond au nom donné à un lien dans la page.	Internet Explorer, Mozilla, Firefox, Opéra.
`length`	Correspond au nombre de liens présents dans la page.	Internet Explorer, Mozilla, Firefox, Opéra.
`target`	Correspond à la fenêtre cible d'un lien (_self, _blank, etc.).	Internet Explorer, Mozilla, Firefox, Opéra.
`text`	Correspond au texte d'un lien présent dans la page.	Mozilla, Firefox, Opéra.
`x`	Correspond à la position horizontale d'un lien présent dans la page.	Mozilla, Firefox, Opéra.
`y`	Correspond à la position verticale d'un lien présent dans la page.	Mozilla, Firefox, Opéra.

Exemple : afficher, dans des boîtes de dialogues, le nombre de liens, le texte et la cible des liens comportant les liens suivants : http://www.editions-eni.fr et http://www.monserveur.com ouverts avec une cible de type _blank pour le premier et de type _self pour le second, le tout au chargement de la page.




```
<html>
<head>
<title>Objet Link</title>
<script language="javascript">
function exemplelinks() {
alert("Il y a : "+document.links.length+" liens sur cette page");
for (i=0;i<document.links.length;i++) {
alert("Voici le texte du lien n° "+(i+1)+" présent dans la page :
"+document.links[i].text);
alert("Voici la cible du lien n° "+(i+1)+" présent dans la page :
"+document.links[i].target);
```

```
}
}
</script>
</head>
<body onLoad="exemplelinks()">
<a href="http://www.editions-eni.fr/" target="_blank">Eni</a><br>
<a href="http://www.form-high-tech.com/" target="_self">Form-High-Tech
</a><br>
</body>
</html>
```

Autres objets utiles

1. regexp

Cet objet permet de manipuler les expressions régulières. Les expressions régulières sont présentes dans la plupart des langages de programmation (d'ailleurs, la syntaxe des expressions régulières en JavaScript est proche de celle du Perl). Elles permettent d'effectuer de nombreux traitements sur les chaînes de caractères, du plus simple au plus élaboré. Avec elles, il est possible de chercher, remplacer, découper des chaînes de caractères. Les maîtriser constitue donc, un réel atout même si leur approche peut sembler fastidieuse au premier abord. Il convient, toutefois, de noter que les expressions régulières peuvent poser problème avec certains navigateurs, tel Internet Explorer ou Safari par exemple. Cette limitation ne remet pas en cause la puissance et la réelle efficacité de celles-ci. Leur principe de fonctionnement consiste à rédiger un masque (pattern en anglais) qu'il est possible d'appliquer à une chaîne de caractères pour la filtrer ou la gérer. Pour utiliser les expressions régulières, il faut d'abord créer un objet de type RegExp, la syntaxe à suivre est la suivante :

```
var monexpression=new RegExp(motif, option) ;
```

Où motif représente le masque de recherche et option correspond à un commutateur. Il peut prendre quatre valeurs différentes, en fonction de ce que l'on désire (par exemple, prendre en compte la casse des caractères). Le tableau suivant, liste les options disponibles :

Caractère d'option	Fonction
" "	Aucune option définie.
"g"	Force une recherche globale.
"i"	Ne tient pas compte de la casse des caractères.
"gi"	Associe les options i et g.

Une autre méthode de création peut également être employée, en suivant la syntaxe :

```
Var monexpression=/motif/option ;
```

Pour rédiger le masque de l'expression régulière, mis à part les caractères classiques, le développeur dispose d'une série de caractères « outils », qu'il est possible de classer en catégories en fonction de leur rôle.

a. Les caractères d'ensemble

Ils permettent de définir une collection de caractères qui devra, selon les cas, apparaître ou non.

Caractère outil	Fonction
[xyz]	Correspond à un ensemble de caractères (ici xyz), placé entre les crochets.
[x-z]	Correspond à un ensemble de caractères en minuscules entre x et z.
[X-Z]	Correspond à un ensemble de caractères en majuscules entre X et Z.
[0-9]	Correspond à un ensemble de caractères entre 0 et 9.
[^xz]	Interdit les caractères suivants ^ (ici x et z).
\d	Correspond à un chiffre. Équivalent à [0-9].
\D	Interdit les chiffres de 0 à 9. Équivalent à [^0-9].

b. Les caractères de groupement

Caractère outil	Fonction
()	Permet de grouper des caractères formant alors un sous-motif.

c. Les caractères de répétition

Ces caractères permettent de tester le nombre d'occurrences d'un caractère.

<table>
<tr><th>Caractère outil</th><th>Fonction</th></tr>
<tr><td>*</td><td>Le caractère peut apparaître un nombre indéfini de fois.</td></tr>
<tr><td>+</td><td>Le caractère doit apparaître au moins une fois.</td></tr>
<tr><td>?</td><td>Le caractère doit apparaître zéro ou une fois.</td></tr>
<tr><td>{x}</td><td>Le caractère doit apparaître le nombre de fois équivalent à x.</td></tr>
<tr><td>{x,z}</td><td>Le caractère doit apparaître au moins x fois et au plus z fois.</td></tr>
<tr><td>x | z</td><td>Le caractère peut être x ou z.</td></tr>
</table>

d. Les caractères de positionnement

Caractère outil	Fonction
^	Précise le début de l'expression dans la chaîne de caractères.
$	Précise la fin de l'expression dans la chaîne de caractères.
\b	Précise le début de mot.
\B	Précise la fin de mot.
(x)	Trouve la chaîne et retient sa position.
X(?=y)	Trouve la chaîne uniquement si x est suivi de y.
X(?!y)	Trouve la chaîne uniquement si x n'est pas suivi de y.

e. Le caractère de choix

Il permet de faire un choix dans l'expression régulière entre plusieurs sous-motifs.

Caractère outil	Fonction

x | z	Le caractère peut être x ou z.

f. Les caractères spéciaux

Caractère outil	Fonction
.	Correspond à tout caractère.
\	Indique que le caractère suivant n'est pas spécial.
\f	Correspond à un saut de page.
\n	Correspond à un saut de ligne.
\r	Correspond à un retour chariot.
\t	Correspond à une tabulation.

g. Les propriétés

Propriété	Résultat	Reconnu par
`lastIndex`	Indique l'indice à partir duquel la recherche suivante doit s'effectuer.	Mozilla, Firefox, Opéra.
`source`	Correspond au texte du masque.	Mozilla, Firefox, Opéra.
`global`	Indique si la recherche doit s'arrêter à la première occurrence trouvée. = « g »	Mozilla, Firefox, Opéra.
`ignoreCase`	Indique si la casse doit être ignorée. = « i ».	Mozilla, Firefox, Opéra.

h. Les méthodes

Méthode	Résultat	Reconnu par
`test()`	Retourne une valeur booléenne si le test est vérifié ou non.	Internet Explorer, Mozilla, Firefox, Opéra.
`exec()`	Retourne la première occurrence trouvée dans la chaîne.	Internet Explorer, Mozilla, Firefox, Opéra.
`match()`	Trouve les occurrences d'une sous-chaîne de caractères.	Internet Explorer, Mozilla, Firefox, Opéra.
`split()`	Trouve des éléments placés entre des séparateurs d'une chaîne de caractères.	Mozilla, Firefox, Opéra.
`toSource()`	Renvoie une déclaration d'objet représentant l'objet spécifié.	Mozilla, Firefox, Opéra.
`toString()`	Renvoie une chaîne de caractères correspondant à l'objet concerné.	Mozilla, Firefox, Opéra.

Cette méthode, bien utile, permet de savoir si une chaîne de caractères correspond bien au masque de l'expression régulière. Elle renvoie true si c'est le cas et false dans le cas contraire.

Les motifs se construisent en utilisant les caractères outils vus précédemment, en fonction du résultat à obtenir. Un motif se rédige de la gauche vers la droite comme dans une phrase classique. Même si cela semble à première vue obscur, un exemple de syntaxe d'une expression régulière pourrait être :

```
"[0-9]{2}[-][a-z]{3}[-][0-9]{4}","gi"
```

Cette expression régulière permet de contrôler la validité d'une chaîne de caractères représentant une date au format 01-jan-2008.

Pour utiliser par la suite un motif, il est nécessaire de construire une variable à l'aide de l'objet RegExp et d'utiliser les méthodes souhaitées.

i. Utilisation de la méthode test()

Cette méthode permet de savoir si une chaîne de caractères correspond au motif. Elle renvoie true si c'est le cas et false dans le cas contraire.

Par exemple, pour tester si une valeur saisie dans une boîte de dialogue correspond à la nouvelle norme européenne de plaque minéralogique, qui sera composée d'une série de caractères formés de trois blocs séparés par des tirets (deux lettres, tiret, trois chiffres maximum, tiret, puis deux lettres), il convient de passer par trois lignes d'instructions correspondants à trois étapes :

Déclaration de la variable correspondant au motif élaboré à partir de l'objet RegExp.

Récupération de la variable saisie dans une boîte de dialogue.

Application de la méthode test() pour vérifier la validité de la saisie.

La syntaxe du masque sera alors la suivante :

```
[A-Z]{2}[-][1-9]{1,3}[-][A-Z]{2}
```

`[A-Z]{2}` correspond à deux lettres obligatoires.

`[-]` correspond au tiret entre le premier et le second bloc de caractères.

`[1-9]{1,3}` indique qu'il faut entre 1 et 3 chiffres compris chacun entre 1 et 9.

`[-]` correspond au tiret entre le second et le troisième bloc de caractères.

`[A-Z]{2}` correspond à la dernière série de lettres.




```
<script language="javascript">
var plaque=new RegExp("[A-Z]{2}[-][1-9]{1,3}[-][A-Z]{2}","g");
var saisieplaque=prompt("Saisissez un numéro de plaque minéralogique
européenne sous la forme 2 lettres-3 chiffres maximum-2 lettres : ");
alert(plaque.test(saisieplaque));
</script>
```

La réponse s'affichera en indiquant la valeur true ou false, si le test de l'expression régulière est respecté ou non.

Exemple : afficher un message indiquant si oui ou non, la valeur saisie dans une boîte de dialogue respecte bien le masque d'une expression régulière, correspondant à un numéro de téléphone français (cinq séries de deux chiffres séparés par des points).

```
<script language="javascript">
var telephone=new RegExp("[0]{1}[1-4]{1}[.][0-9]{2}[.][0-9]{2}[.][0-
9]{2}[.][0-9]{2}","g");
var saisienumero=prompt("Saisissez un numéro de téléphone français
sous la forme 00.00.00.00.00 : ");
alert(telephone.test(saisienumero));
</script>
```

Petite particularité ici, où le premier chiffre ne peut être qu'un zéro, mais doit être saisi tout de même.

j. Utilisation de la méthode exec()

Elle retourne la première occurrence (et seulement celle-la) trouvée, correspondant au motif de l'expression régulière dans une liste de données. La syntaxe est la suivante :

`variableexpressionrégulière.exec(listededonnées) ;`

Exemple : afficher le premier numéro de téléphone présent d'un agenda, respectant le masque[0]{1}[1-4]{1}[.][0-9]{2}[.][0-9]{2}[.][0-9]{2}[.][0-9]{2}

```
<script language="javascript">
var telephone=new RegExp("[0]{1}[1-4]{1}[.][0-9]{2}[.][0-9]{2}[.][0-
9]{2}[.][0-9]{2}","g");
var agenda=new Array ("03.44.12.230","01.000.01.01.01","01.12.13.14.15");
var resultat = telephone.exec(agenda);
alert("Le numéro de téléphone correspondant au masque est le suivant :
"+resultat);
</script>
```

k. Utilisation de la méthode match()

Elle permet de trouver toutes les occurrences correspondant au motif d'une expression régulière dans une variable de type String. La syntaxe est la suivante :

`Variabletexte.match(variableexpressionrégulière) ;`

Exemple : afficher dans une boîte de dialogue les numéros de téléphone correspondants au motif déjà utilisé auparavant :



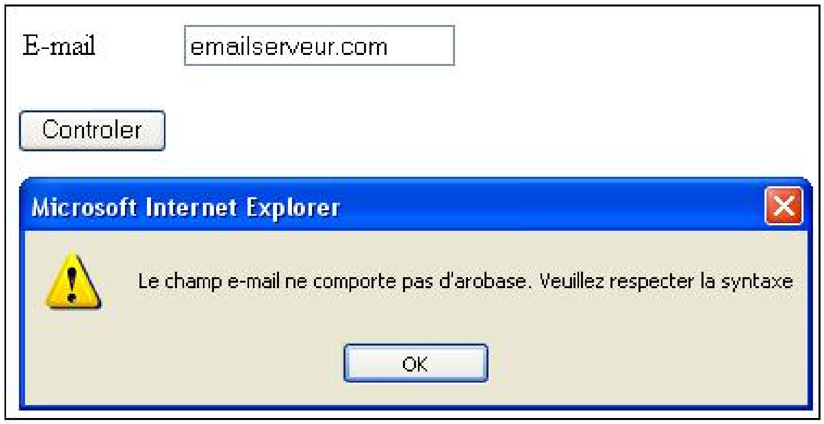

```
<script language="javascript">
var telephone=new RegExp("[0]{1}[1-4]{1}[.][0-9]{2}[.][0-9]{2}[.][0-
9]{2}[.][0-9]{2}","g");
var agenda= ("03.44.12.230 01.000.01.01.01 01.12.13.14.15
01.12.13.14.16");
var resultat = agenda.match(telephone);
var nbelements=resultat.length;
message=nbelements+" numéros trouvés correspondants au masque :";
for (i=0;i<nbelements;i++) {
var reponse=resultat[i];
message=message+"\r"+reponse;
}
alert(message);
</script>
```

Attention "\r " ne fonctionne pas avec Firefox.

l. Utilisation de la méthode search()

La méthode search() s’applique aux expressions régulières, comme elle s’applique aux variables textes.

Exemple : contrôler la présence d’une arobase dans un champ de formulaire et afficher un message d’avertissement ou de conformité en fonction du résultat.




```
<html>
<head>
<title>Objet RegExp - Méthode search</title>
<script language="javascript">
function controlemail() {
var email=document.form1.monmail.value;
var resultat=email.search("@");
if (resultat!=false) {
alert("Le champ e-mail ne comporte pas d’arobase. Veuillez respecter
la syntaxe");
}
else {
alert("Merci, votre adresse mail comporte une arobase");
}
}
</script>
</head>
<body>
<form name="form1" method="post" action="">
  <p> </p>
  <table width="400" border="0">
    <tr>
      <td>E-mail</td>
      <td><input name="monmail" type="text" id="monmail"
value="Veuillez saisir votre e-mail personnel ici"></td>
    </tr>
  </table>
  <p>
    <input type="button" name="Submit" value="Controler"
onClick="controlemail()">
</p>
</form>
</body>
</html>
```

m. Utilisation de la méthode replace()

Cette méthode permet de trouver et de remplacer une valeur par une autre, selon un motif défini dans l’expression régulière.

La syntaxe de la méthode replace() est la suivante :

```
chaîneàinspecter.replace(chaîneàremplacer,chaînederemplacement) ;
```

Exemple : afficher, dans la page, l'agenda modifié par remplacement des caractères slash et double-point, par un simple point comme caractère de délimitation.

```
<script language="javascript">
var telephone=new RegExp("[/:]","g");
var agenda= ("03/44/12/23 01/00/01/01/01 01:12:13:14:15");
var resultat=agenda.replace(telephone,".");
document.write(resultat);
</script>
```

En fait, les expressions régulières, même si elles demandent un peu de pratique, sont utilisées dans beaucoup de scripts concernant la validation, la recherche ou la modification de chaînes de caractères.

2. Math

Cet objet permet d'effectuer des calculs complexes. Il dispose d'un certain nombre de propriétés et de méthodes.

a. Les propriétés

Propriété	Résultat	Reconnu par
E	Renvoie la constante d'Euler dont la valeur est proche de 2,718.	Internet Explorer, Mozilla, Firefox, Opéra.
LN2	Renvoie le logarithme naturel de 2.	Internet Explorer, Mozilla, Firefox, Opéra.
LN10	Renvoie le logarithme naturel de 10.	Internet Explorer, Mozilla, Firefox, Opéra.
LOG2E	Renvoie le logarithme de 2.	Internet Explorer, Mozilla, Firefox, Opéra.
LOG10E	Renvoie le logarithme de 10.	Internet Explorer, Mozilla, Firefox, Opéra.
PI	Renvoie la valeur de Pi soit approximativement 3.14159.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : afficher le résultat du calcul d'une circonférence d'un cercle à partir de son diamètre.





```
<script language="javascript">
var diametre=prompt("Quel est le diametre du cercle ? :");
var circonference=diametre*Math.PI;
alert("Voici la circonférence du cercle :"+circonference);
</script>
```

b. Les méthodes

Méthode	Résultat	Reconnu par
`abs()`	Extrait la valeur positive absolue d'un nombre passé en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`acos()`	Calcule l'angle dont l'argument représente le cosinus.	Internet Explorer, Mozilla, Firefox, Opéra.
`asin()`	Calcule l'angle dont l'argument représente le sinus.	Internet Explorer, Mozilla, Firefox, Opéra.
`atan()`	Calcule l'angle dont l'argument représente la tangente.	Internet Explorer, Mozilla, Firefox, Opéra.
`ceil()`	Renvoie un nombre correspondant à l'entier supérieur d'une valeur passée en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`floor()`	Renvoie un nombre correspondant à l'entier inférieur d'une valeur passée en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`log()`	Calcule le logarithme népérien d'un nombre passé en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`max()`	Renvoie le nombre le plus grand dans la série, passée en arguments.	Internet Explorer, Mozilla, Firefox, Opéra.
`min()`	Renvoie le nombre le plus petit dans la série, passée en arguments.	Internet Explorer, Mozilla, Firefox, Opéra.
`pow()`	Calcule le résultat d'un nombre élevé à une puissance, passée en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`random()`	Renvoie un nombre aléatoire compris entre 0 et 1.	Internet Explorer, Mozilla, Firefox, Opéra.
`round()`	Arrondit un nombre à l'entier le plus proche.	Internet Explorer, Mozilla, Firefox, Opéra.
`sqrt()`	Calcule la racine carrée d'un nombre, passé en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`sin()`	Calcule le sinus d'un nombre, passé en argument.	Internet Explorer, Mozilla, Firefox, Opéra.
`tan()`	Calcule la tangente d'un angle, passé en argument.	Internet Explorer, Mozilla, Firefox, Opéra.

Exemple : créer une application en JavaScript permettant d'effectuer des tirages de la loterie nationale (c'est-à-dire avec 6 chiffres distincts de 1 à 49 et un numéro complémentaire). Attention, un numéro ne peut apparaître qu'une seule fois.




```
<html>
<head>
<title>Objets Array et Math</title>
<script language="JavaScript">
function tirage6numeros() {
var tirage=new Array(12);
var tiragetrie=new Array(12);
var bontirage=new Array();
for(i=1;i<12;i++) {
boule=Math.ceil(Math.random()*49);
tirage[i]=boule;
}
tiragetrie=tirage.sort(function(n,m){return n-m});
for(i=1;i<12;i++) {
if( tiragetrie[i]!=tiragetrie[i+1]) {
bontirage[i]=tiragetrie[i];
}
}
bontiragetrie=bontirage.sort(function(n,m){return n-m});
for(i=1;i<7;i++) {
nomchamp="boule"+i;
instruction="document.form1."+nomchamp+".value=bontiragetrie[i]";
eval(instruction);
}
}
function fcomplementaire(){
numcomp=Math.ceil(Math.random()*49);
document.form1.complementaire.value=numcomp;
}
</script>
</head>
<body>
</center>
<form name="form1" method="post" action="">
  <table width="64%" border="1" align="center">
    <tr>
      <td><div align="center">Boule 1 </div></td>
      <td><div align="center">Boule 2 </div></td>
      <td><div align="center">Boule 3 </div></td>
      <td><div align="center">Boule 4 </div></td>
      <td><div align="center">Boule 5 </div></td>
      <td><div align="center">Boule 6 </div></td>
    </tr>
    <tr>
      <td><div align="center">
        <input name="boule1" type="text" id="1" size="5"
maxlength="5">
      </div></td>
      <td><div align="center">
        <input name="boule2" type="text" id="2" size="5"
maxlength="5">
      </div></td>
      <td><div align="center">
        <input name="boule3" type="text" id="3" size="5"
maxlength="5">
      </div></td>
      <td><div align="center">
        <input name="boule4" type="text" id="4" size="5"
maxlength="5">
      </div></td>
      <td><div align="center">
        <input name="boule5" type="text" id="5" size="5"
maxlength="5">
      </div></td>
      <td><div align="center">
```

```
        <input name="boule6" type="text" id="6" size="5"
maxlength="5">
      </div></td>
    </tr>
    <tr>
      <td colspan="6"><div align="center">
        <input type="button" name="Submit" value="Tirage 6
num&eacute;ros" onClick="tirage6numeros()">
      </div></td>
    </tr>
  </table>
  <p> </p>
  <table width="13%" border="1" align="center">
    <tr>
      <td><div align="center">Num&eacute;ro compl&eacute;mentaire
</div></td>
    </tr>
    <tr>
      <td><div align="center">
        <input name="complementaire" type="text" id="complementaire"
size="5" maxlength="5">
      </div></td>
    </tr>
    <tr>
      <td><div align="center">
        <input type="button" name="Submit2" value="Tirage
complementaire" onClick="fcomplementaire()">
      </div></td>
    </tr>
  </table>
  <p> </p>
  <p> </p>
</form>
</body>
</html>
```

Dans ce script, le tirage des six premiers numéros et du numéro complémentaire sont dissociés. Ce script présente plusieurs difficultés. La première consiste à obtenir six numéros dans la limite fixée (1 à 49). La seconde constitue le cœur de la fonction puisqu'il est impossible de retrouver deux fois le même chiffre dans le même tirage. C'est la fonction tirage6numeros qui lance le traitement du tirage des six premiers numéros, de manière simultanée. Ainsi, il n'est pas nécessaire de cliquer six fois sur un bouton. Pour traiter ce tirage, il est nécessaire de passer par l'intermédiaire de tableaux. En effet, l'usage des différentes méthodes de traitement (tri, suppression), impose la création de plusieurs tableaux. Ce sont donc, les tableaux tirage, tiragetrie et bontirage qui stockent les données durant ces différentes étapes. Les deux premiers disposent d'un nombre d'éléments déterminés (12), puisqu'il faut effectuer plus de tirages qu'il y a de numéros à sortir (tenir compte des suppressions de doublon). Dans l'exemple, le nombre de tirages de numéros est fixé à 12, ce qui semble suffisant car il est rare d'obtenir 6 doublons. Au final, seul six numéros sont affichés, les autres (s'il y en a) ne sont pas affichés. Pour compléter le premier tableau, l'utilisation de la méthode random de l'objet Math est intéressante. Rappelons que cette méthode renvoie un nombre entre 0 et 1. Ainsi, pour obtenir un nombre compris entre 1 et 49, il suffit de le multiplier par 49. Seul subsiste le problème d'un résultat égal à zéro, il sera réglé par l'emploi de la méthode ceil permettant d'extraire l'entier supérieur. Ainsi, même si le résultat donné par la méthode random() est équivalent à 0.7365456, par exemple, c'est bien un qui est retourné. La suite du script permet d'alimenter le tableau tiragetrie, comprenant les numéros triés. L'enjeu réside dans le fait, qu'il faut obtenir un tableau qui ne comporte pas de doublons. Après le tri, les doublons sont côte à côte dans le tableau, ce qui facilite leur suppression. La méthode sort() est, ici, employée conjointement avec une fonction de comparaison placée entre parenthèses, qui permet de renvoyer des éléments numériques triés. Sans cette fonction de comparaison, les éléments seraient triés dans l'ordre lexicographique. Une fois les éléments triés, il suffit de les comparer un à un puis de les transférer dans le tableau nommé bontirage. Ce dernier tableau peut alors comprendre des éléments avec une valeur undefined. En triant le tableau, ces éléments sont placés à la fin et n'apparaîtront donc pas. Il reste à afficher les éléments du tableau dans les champs du formulaire. Pour y parvenir, il est possible d'utiliser deux méthodes. La première consiste à employer une méthode utilisant le DOM. Celle-ci sera détaillée au chapitre Améliorer l'interactivité avec JavaScript et CSS. La seconde consiste à faire une boucle et d'utiliser le compteur i pour compléter l'instruction nomchamp="boule"+i;. Cette expression est ensuite traitée par l'intermédiaire de la méthode eval(). Ainsi, la variable texte nomchamp est constituée de la chaîne de caractères « boule » à laquelle il suffit d'ajouter le numéro du compteur i. Il faut ensuite construire une instruction alternant chaîne de caractères et la variable nomchamp afin qu'elle soit exécutée par le mot clé eval(). La seconde fonction permet de traiter le numéro complémentaire par la même méthode.

3. frame

L'objet frame correspond aux cadres (frames) présents dans une page HTML. L'objet frame dispose d'une propriété

essentielle.

a. La propriété

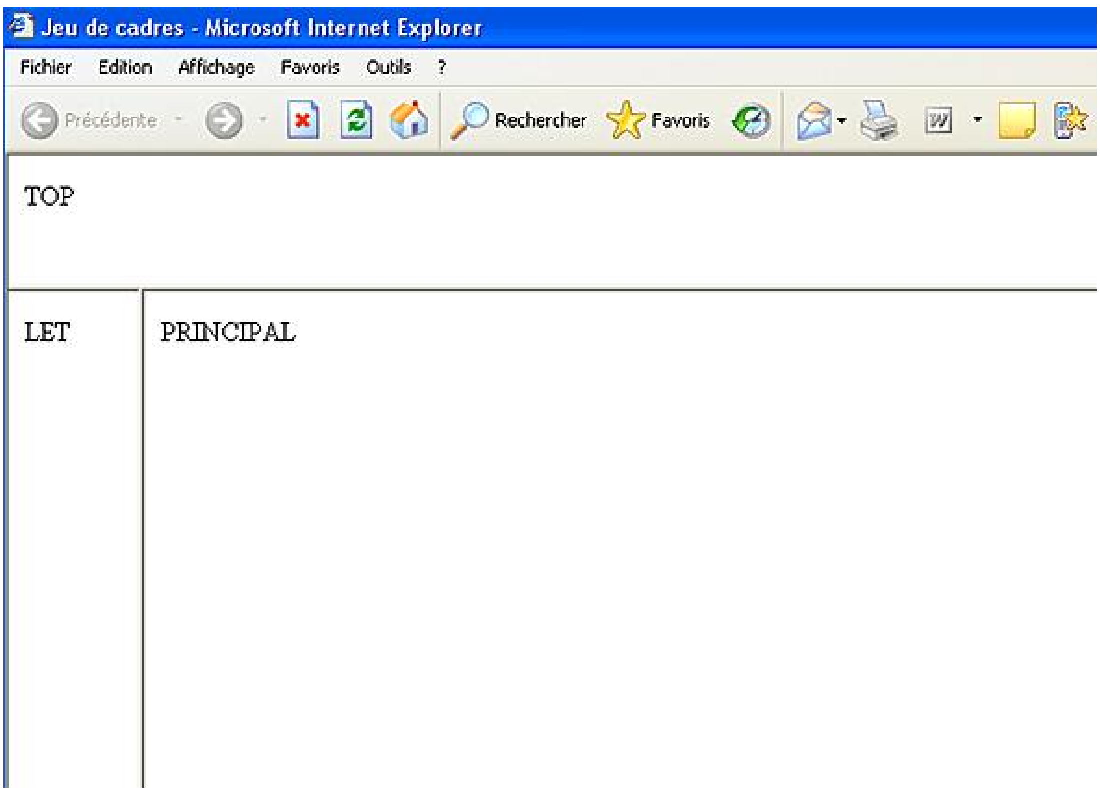
Propriété	Résultat	Reconnu par
`length`	Correspond au nombre de frames présents dans la page.	Internet Explorer, Mozilla, Firefox, Opéra.

En fait, l'objet frame est très proche de l'objet Window et toutes les propriétés et méthodes dont nous avons déjà parlées pour ce dernier, s'appliquent également à l'objet frame. Avec frame, il est possible d'accéder à une fenêtre particulière du jeu de cadres et deux méthodes sont alors possibles.

La première consiste à utiliser les numéros d'indice de chaque élément du jeu de cadres. Ainsi le premier indice (le zéro comme toujours) correspond au cadre défini le premier, le un correspond au second…

La seconde méthode (et sans doute la meilleure) consiste à identifier les cadres par le nom, qui leur a été attribué avec l'attribut name.

Soit le jeu de cadres suivant :

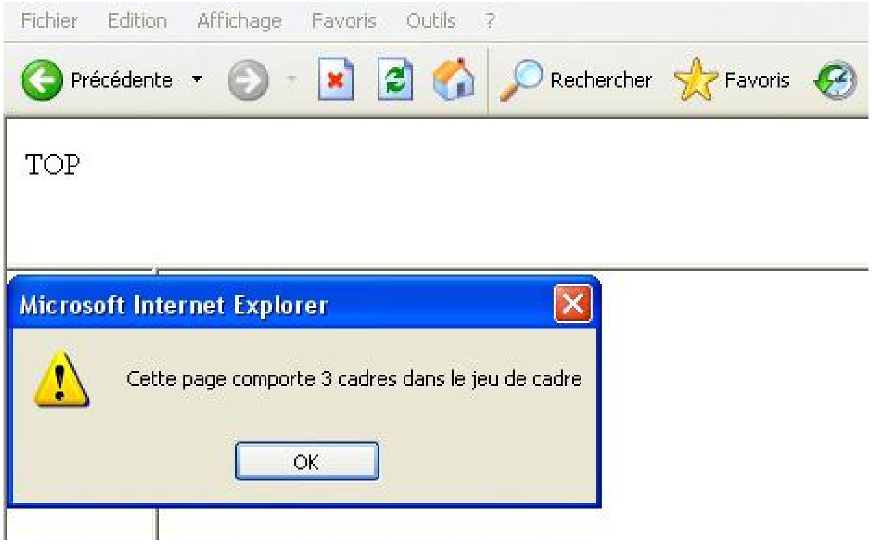


Le code HTML correspondant est :

```
<frameset rows="80,*" cols="*" frameborder="yes" border="1" frame-
spacing="0">
  <frame src="top.html" name="topFrame" scrolling="No" nore-
size="noresize" id="topFrame" title="Haut" />
  <frameset cols="80,*" frameborder="yes" border="1"
framespacing="0">
    <frame src="lef.html" name="leftFrame" scrolling="No" nore-
size="noresize" id="leftFrame" title="Gauche" />
    <frame src="principal.html" name="mainFrame" id="mainFrame"
title="Principal" />
  </frameset>
</frameset>
```

Ainsi, les cadres topFrame, leftFrame et mainFrame peuvent recevoir par JavaScript le contenu d'une page HTML.

Exemple : afficher le nombre et le nom des cadres composant le jeu de cadres décrit ci-dessus, puis charger une autre page dans le cadre left. Le script sera placé dans la page dénommée principal.




```
<script language="javascript">
alert("Cette page comporte "+parent.frames.length+" cadres
dans le jeu de cadres");
for(var i=0; i < parent.frames.length; i++) {
alert(parent.frames[i].name);
parent.leftFrame.location.href = "autrepage.html";
}
</script>
```

Ce script doit être placé dans une des pages du jeu de cadres, d'où le recours au préfixe parent pour désigner le jeu de cadres complet.

4. Event

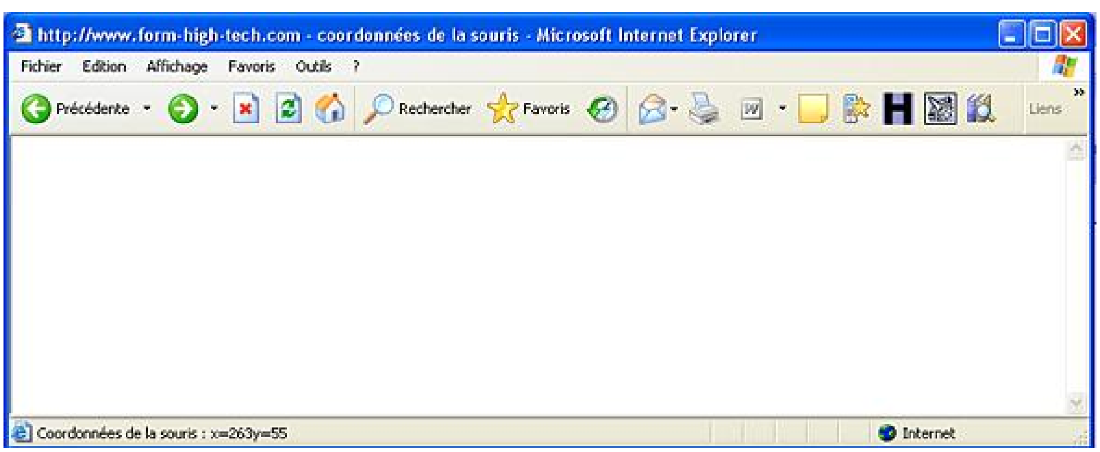
Il s'agit d'un objet particulier qui permet de surveiller les évènements pouvant, lors de la consultation de la page, survenir soit par le biais de la souris, soit par celui du clavier. Le gros inconvénient de l'utilisation de cet objet est qu'il n'est pas reconnu de la même manière par tous les navigateurs. D'ailleurs, certaines propriétés n'existent pas sous Internet Explorer.

a. Les propriétés

Propriété	Résultat	Reconnu par
altKey, ctrlKey, shiftKey	Contrôle si des touches ont été enfoncées au même instant qu'une pression sur la touche [Alt], [Ctrl] ou [Shift].	Internet Explorer
clientX, clientY	Retourne l'abscisse ou l'ordonnée de l'évènement par rapport au document.	Internet Explorer
keyCode	Retourne le code de la touche qui vient d'être enfoncée.	Internet Explorer
height, width	Contrôle si les dimensions en hauteur de la fenêtre	Mozilla, Firefox, Opéra.

	ont été modifiées.	
`layerX, layerY`	Contrôle si la position horizontale ou verticale de la souris a changé.	Mozilla, Firefox, Opéra.
`offsetX, offsetY`	Représente la position horizontale ou verticale de la souris	Internet Explorer
`pageX, pageY`	Représente la position horizontale ou verticale de la souris dans la fenêtre correspondant au document HTML.	Mozilla, Firefox, Opéra.
`screenX, screenY`	Représente la position horizontale ou verticale de la souris dans la fenêtre du navigateur.	Mozilla, Firefox, Opéra.
`which`	Retourne la touche ou le bouton à l'origine de l'évènement.	Mozilla, Firefox, Opéra.
`type`	Retourne le nom de l'évènement qui vient de se produire.	Mozilla, Firefox, Opéra.
`x, y`	Correspond aux coordonnées de la position de la souris ou à la nouvelle position de la fenêtre en cas de redimensionnement.	Internet Explorer

Exemple : afficher les coordonnées de la souris dans la barre des status :




```
<html>
<head>
<title>coordonnées de la souris</title>
<script language="javascript">
function souris(event){
var posX = event.clientX;
var posY = event.clientY;
window.status="Coordonnées de la souris : x="+posX+"y="+posY;
}
```

```
</script>
</head>
<body onmousemove="souris(event);">
</body>
</html>
```

Le script s'exécute lors d'un mouvement de la souris. L'objet event, associé aux méthodes clientX et clientY, permet de déterminer les variables posX et posY puis de compléter la barre de statut.

Attention ce script ne fonctionne que sous Internet Explorer.

JavaScript et les cookies

Contrairement à ce que l'on pourrait penser, cette partie de l'ouvrage n'est pas destinée à vous transmettre quelques informations de cuisine ou des recettes. Les cookies (dont la traduction pourrait être témoins) sont, simplement, de petits fichiers qui sont laissés sur le disque dur du visiteur, par le navigateur. La taille de ces fichiers est très limitée (inférieure à 5 Ko) mais stockent différentes informations concernant le visiteur comme, par exemple, son identifiant (s'il le désire), ou l'objet de ses précédentes visites sur le site. Les cookies sont, parfois, mal perçus par les visiteurs et ces derniers sont, alors, tentés de les interdire en paramétrant leur navigateur. Il est vrai que leur usage permet de suivre quasiment à la trace les visiteurs lorsqu'ils sont utilisés à des fins malveillantes. À l'inverse, les cookies ne sont pas des virus et n'en favorisent pas non plus la diffusion. JavaScript permet de gérer facilement ces cookies.

1. Principe et utilisation des cookies

Les cookies sont de petits fichiers au format txt (et contiennent donc du texte) stockés, différemment selon le navigateur employé, sur le disque dur. Avec Internet Explorer, les cookies sont stockés dans le répertoire c:\Documents and Settings\utilisateur\Cookies sous la forme de plusieurs fichiers dont le nom peut permettre d'en deviner la provenance. Vous serez certainement surpris et effrayé du nombre de fichiers ainsi stockés sur votre ordinateur, à votre insu. Avec Firefox/Mozilla, les cookies sont stockés dans un fichier unique (cookies.txt), et avec avec Opera, ils sont, en plus de cela, cryptés.

Concrètement, le principe de fonctionnement des cookies est très simple. Lors de la première visite d'un visiteur, le navigateur écrit dans un fichier particulier des informations, qu'il lui sera possible de lire lors de la prochaine connexion du visiteur. Les utilisations possibles sont très variées et vont de la navigation avancée jusqu'à la gestion des caddies de commande en ligne, en passant par les compteurs de visite. Par exemple, il est ainsi très facile de stocker le nom, le prénom ou la langue de préférence du visiteur, afin de personnaliser son interface lorsqu'il se connectera de nouveau et de lui proposer des pages dans sa langue préférée. Les cookies peuvent être modifiés et contenir des valeurs modifiables ultérieurement, et une fois l'utilité des cookies passée, il est possible de les supprimer.

Afin de tester convenablement les scripts comprenant des cookies, il convient de régler le niveau de sécurité du navigateur. En effet, comme indiqué précédemment, même si un script est correct, le navigateur ne pourra pas écrire avec un réglage sur un niveau élevé de sécurité. Il est, donc, important d'effectuer un paramétrage des navigateurs.

2. Paramétrage des navigateurs

Lorsque les cookies sont employés pour faciliter l'authentification, ils peuvent stocker des informations confidentielles (identifiant, mot de passe, etc.). Cet aspect pourrait paraître fort inquiétant. En effet, rien n'interdit de penser qu'un utilisateur malveillant puisse récupérer des informations provenant d'autres domaines. Heureusement, il est impossible de lire un cookie n'appartenant pas au domaine de la page ayant déposé celui-ci. Ainsi, l'auteur du script ne peut lire que les cookies qu'il a déposés et non pas, ceux laissés par d'autres. De toute manière, si les cookies ne sont pas appréciés par les visiteurs, ces derniers pourront toujours les désactiver dans leur navigateur.

Tous les navigateurs sont capables d'effacer les cookies présent sur l'ordinateur. Avec Firefox/Mozilla, la commande **Effacer mes traces** du menu **Outils** permet de vider le contenu des cookies. Avec Internet Explorer, il faut cliquer sur le bouton **Effacer les cookies** accessible par le Menu **Outils/Options Internet**. Pour interdire l'écriture des cookies sur l'ordinateur avec Firefox/Mozilla, il suffit de se rendre dans le menu **Outils** puis de choisir **Options** et enfin de décocher la case **Accepter les cookies** de l'onglet **Vie privée**.

Avec Internet Explorer, il faut faire glisser vers le haut, le curseur de l'onglet **Confidentialité** de la boîte de dialogue **Options Internet**, visible par le menu **Outils**. À l'inverse, pour les prochains exemples, il est important d'accepter les cookies.

3. Limites d'utilisation des cookies

Les exemples précédents d'utilisation montrent clairement l'intérêt, mais aussi les limites de l'utilisation des cookies. Inutile d'espérer pouvoir créer un compteur de visites avec JavaScript et les cookies, le seul moyen d'y parvenir est de passer par un langage de programmation côté serveur, tel ASP ou PHP. Certains sites nécessitent l'utilisation des cookies pour fonctionner correctement et les avantages de l'utilisation de ceux-ci, même s'ils sont intéressants, notamment, pour la personnalisation de l'interface des sites, sont souvent décriés, d'autant plus que des problèmes de sécurité ont sérieusement limité leur popularité.

4. Application des cookies

L'étape fondamentale consiste à créer un cookie sur le poste du visiteur. Cette création nécessite la présence d'un certain nombre d'informations obligatoires.

5. Les informations obligatoires pour la création d'un cookie

L'accès aux cookies s'effectue par la propriété cookie de l'objet document. Il existe deux types de fonctionnement des cookies, l'un temporaire (dans le cas où, il n'est pas nécessaire de stocker des informations pour un usage ultérieur), et l'autre permanent (lorsque les cookies restent présents après la connexion). Pour un usage temporaire, il suffit d'indiquer le nom du cookie à créer et les informations à y stocker. Par contre, pour un usage permanent, il est nécessaire d'indiquer un nom, une date d'expiration (durée de vie), une valeur et un nom de domaine, pour le reconnaître parmi tous les autres. De plus, la taille des cookies ne doit pas être supérieure à 5 Ko, il est impossible d'en stocker plus de vingt par domaine et le nombre total de cookies, stockés sur l'ordinateur, ne doit pas être supérieur à 300. Les informations sont stockées sous la forme d'une chaîne de caractères sans espace. Cet aspect représente la principale difficulté de la gestion des cookies. Pour une utilisation temporaire, la syntaxe permettant la création d'un cookie est la suivante :

```
document.cookie="nomducookie=valeurducookie" ;
```

Exemple : créer un cookie temporaire puis l'afficher dans une boîte de dialogue.




```
<script language="javascript">
document.cookie="moncookie=mon_nom";
alert(document.cookie);
</script>
```

Cet exemple démontre la facilité de création des cookies temporaires. Par contre, l'intérêt d'utilisation des cookies temporaires reste limité, puisque les informations ne peuvent être stockées pour un usage ultérieur. En effet, en fermant la fenêtre du navigateur le cookie est automatiquement supprimé. Si vous désirez conserver des informations, il vous faut utiliser les cookies permanents.

6. Les cookies permanents

Il ne faut pas se laisser tromper par le terme de permanent, car les cookies permanents ont également une durée de vie limitée. Celle-ci est fixée par le développeur et elle peut être très brève, ou inversement très longue. En effet, pour créer un cookie permanent, il est nécessaire d'ajouter une information supplémentaire permettant d'indiquer la durée de vie du cookie. En fait, cela correspond à sa date d'expiration que l'on renseigne avec la syntaxe suivante :

```
expires=datedexpiration
```

où datedexpiration peut être une date absolue (par exemple le 31 décembre 2099), ou une date relative par rapport à la date de création (par exemple 100 jours après la date de création du cookie).

```
document.cookie="nomducookie= valeurducookie ;
expires=datedexpiration ;
```

Exemple : créer un cookie permanent avec une date d'expiration correspondant au 31 décembre 2099.

```
<script type="text/javascript" language="JavaScript">
var name="moncookie";
var value="mavaleur";
document.cookie = name + "=" + value + ";expires="+"Sat,31-Dec-2099
00:00:01 GMT";
</script>
```

Tout d'abord, il est nécessaire de créer et d'affecter les variables name et value par une valeur de type texte. Ensuite, il suffit d'utiliser la méthode cookie() de l'objet document et de renseigner les paramètres de création. Le format de la date d'expiration est, ici, exprimé en GMT.

Exemple : créer un cookie permanent dont l'expiration interviendra trente secondes plus tard.




```
<script type="text/javascript" language="JavaScript">
date_expiration=new Date();
var expiration=30000;
date_expiration.setTime(date_expiration.getTime() + (expiration));
document.cookie =moncookie="mavaleur";expires=date_expiration;
alert("Le cookie arrivera à expiration à ce moment :
"+date_expiration);
</script>
```

La première étape vise, ici, à créer un objet de type date() appelé date_expiration. Ensuite, il faut créer une variable expiration correspondant au temps en millisecondes (ici 30000 millisecondes = 30 secondes) qui sera ajouté au moment présent par l'instruction `date_expiration.setTime(date_expiration.getTime() + (expiration));`

Ensuite, il suffit de créer le cookie par la méthode précédente, en ajoutant le paramètre expires correspondant à la date d'expiration. Il est, alors, facile d'afficher, dans une boîte de dialogue, le moment prévu de l'expiration du cookie.

La date d'expiration est, donc, une information très importante pour la création ou la suppression d'un cookie, comme nous le verrons plus tard. Mais d'autres informations peuvent aussi être déclarées lors de la création d'un cookie, comme le domaine, le chemin de stockage du cookie ou encore l'attribut de sécurité. Il suffit simplement de les ajouter à la suite en les séparant par des points-virgules. Mais ces informations, à l'inverse de name, value et expires, sont optionnelles.

7. Ajouter des informations optionnelles au cookie

Les informations optionnelles permettent de mieux identifier le cookie. Elles sont au nombre de trois.

a. Le domaine de validité

Le fait d'ajouter un domaine de validité permet d'identifier les cookies, ce qui est utile notamment lors de la relecture. Si cette option n'est pas spécifiée, le navigateur prendra le nom de domaine de la page contenant le script, ce qui généralement est suffisant. Une fois le domaine spécifié, il est normalement impossible de modifier le contenu du cookie à partir d'une page n'appartenant pas au domaine. Le nombre de cookies autorisés par domaine est limité à vingt. Avec Internet Explorer, le nombre de cookies par domaine est affiché entre crochets (informations visibles dans le répertoire de gestion des cookies). La syntaxe, permettant de spécifier un domaine, est la suivante :

```
domain=nomdudomaine ;
```

b. Le chemin d'accès

Le chemin d'accès permet d'indiquer, pour quelle partie de l'URL, le cookie est valable. Il est possible de lire les cookies laissés par les pages des sur-répertoires et non ceux laissés par les pages des sous-répertoires.

La syntaxe correspondante est :

```
path=chemin ;
```

c. L'attribut de sécurité

L'attribut de sécurité correspond au type de connexion (https et non http). Si vous activez cette option, le cookie ne sera transmis que si la connexion est sécurisée (c'est-à-dire en utilisant le protocole https).

Pour activer la sécurisation, il suffit simplement de faire figurer la mention secure dans les paramètres du cookie, sans paramétrage particulier.

Attention si vous activez l'option secure, le script ne s'exécutera pas, si vous faites un essai avec le protocole http.

Comme indiqué précédemment, ces informations facultatives peuvent s'ajouter facilement aux mentions obligatoires, en les séparant par un point-virgule. Donc, la syntaxe complète de création d'un cookie (sécurisé) est la suivante :

```
document.cookie = "name=value; expires=dateexpiration; path=chemin;
domain=nomdudomaine; secure";
```

8. Lecture d'un cookie

La méthode de lecture d'un cookie est relativement simple, il suffit de manipuler la chaîne de caractères présente dans le cookie, identifié par son nom. La méthode de manipulation peut correspondre à une expression régulière.

Exemple : créer deux pages, l'une permettant d'écrire un cookie comportant l'heure et la date de visite de la page, l'autre permettant de lire le cookie et d'afficher une boîte de dialogue avec la date et l'heure de dernière visite, inscrits dans le cookie.

Script de la page d'écriture du cookie :




```
<script type="text/javascript" language="JavaScript">
var nom="moncookie";
var madate=new Date();
var heure=madate.getHours();
var minute=madate.getMinutes();
var seconde=madate.getSeconds();
var jour=madate.getDay()+1;
var mois=madate.getMonth()+1;
var annee=madate.getFullYear();
var datecomplete=jour+"/"+mois+"/"+annee+"
"+heure+":"+minute+":"+seconde;
var value= datecomplete ;
document.cookie = nom + "=" + value + ";expires="+"Sat,31-Dec-2099
00:00:01 GMT;domain=mondomaine.com";
if (document.cookie.length>0) {
alert("il y a un cookie de déposé");
}
```

Afin de pouvoir écrire l'heure et la date de la visite, il est nécessaire de créer toutes les variables pour y stocker l'ensemble des éléments d'information du temps (année, mois, jour, heure, minute, seconde). Il est également utile de créer une variable, appelée datecomplete, qui concatène l'ensemble des informations provenant des variables de temps. L'étape suivante permet de créer et de paramétrer le cookie composé du nom, de sa valeur correspondant à datecomplete, de la date d'expiration et du domaine concerné. Enfin, en testant document.cookie vous pouvez savoir si le cookie a bien été déposé et si sa longueur est supérieure à zéro. L'affichage d'une boîte de dialogue confirme, alors, le dépôt du cookie.

Script de la page de lecture et d'affichage du cookie :



```
<script language="javascript">
var debut = document.cookie.indexOf("moncookie" );
if( debut == -1 )    {
alert("il n'y a pas de cookie");
}
else {
var fin=document.cookie.length;
alert(document.cookie.substring(debut,fin));
}
</script>
```

La première étape consiste à tester la présence du cookie qui nous intéresse. En l'occurrence, ici, le cookie intéressant se nomme moncookie. Cela s'obtient en effectuant une recherche de la position du nom du cookie dans la chaîne de caractères du cookie. Le résultat est affecté à une variable nommée debut. Cette position permettra, par la suite, de retrouver les informations désirées dans la chaîne. Si le test est égal à -1, cela signifie que le cookie n'existe pas (c'est la réponse de la méthode indexOf() lorsque la recherche n'a pas abouti). Une boîte de dialogue permet de l'indiquer. Dans l'autre cas, le cookie a bien été retrouvé et il s'agit à présent d'en lire le contenu. Pour cela, il faut calculer la longueur de la chaîne de caractères correspondant au cookie et la stocker dans une variable nommée fin. Ensuite, il ne reste plus qu'à demander l'affichage du cookie en sélectionnant le début et la fin de la chaîne de caractères composant le cookie.

9. Mise à jour d'un cookie

La mise à jour s'effectue simplement par la lecture puis l'affectation d'une nouvelle valeur au cookie. Le seul problème réside dans le fait que l'ensemble des informations stockées est dans une chaîne de caractères, ce qui ne facilite pas l'application d'opérations arithmétiques, comme pour faire un compteur de visites de pages, par exemple.

Exemple : afficher un compteur représentant le nombre de fois qu'un visiteur consulte la même page.






```
<html>
<head>
<title>Modifier un cookie</title>
<script type="text/javascript" language="JavaScript">
var fin=document.cookie.length;
if (fin>0) {
alert("Mon cookie est déjà présent");
    var valeurcookie=document.cookie.substring(10,fin);
    var cookienum=parseInt(valeurcookie);
    cookienum=cookienum+1;
    alert("C'est votre "+cookienum+" ème visite sur cette page");
    document.cookie="moncookie="+cookienum+ ";expires="+"Sat,31-Dec-2099
00:00:01 GMT;domain=www.mondomaine.com;path=/";
}
else {
alert("Mon cookie n'est pas présent et c'est votre première visite
sur cette page");
var nom="moncookie";
var value=1;
document.cookie = nom + "=" + value + ";expires="+"Sat,31-Dec-2099
```

```
00:00:01 GMT;domain=mondomaine.com;path=/";
var debut = document.cookie.indexOf("moncookie" );
}
</script>
</head>
<body>
</body>
</html>
```

Attention ! Ce compteur n’est pas un vrai compteur de visites. Il faut le distinguer du compteur de visites prenant en compte toutes les visites de tous les visiteurs. Pour ce genre de compteur, il faut utiliser des langages avec programmation, côté serveur (le compteur étant stocké sur le serveur, il peut prendre en compte les visites provenant de visiteurs différents). Ici, par contre, le compteur ne s’intéresse qu’aux visites du visiteur connecté sur la page puisqu’il prend les informations stockées sur l’ordinateur, par l’intermédiaire des cookies.

Comme pour la lecture, il faut d’abord tester la présence du cookie. Ici, il suffit de calculer le nombre de caractères du cookie et de vérifier que ce nombre est supérieur à zéro. Si c’est le cas, vous affichez une boîte de dialogue informant de la présence du cookie. Le travail suivant consiste à définir le début et la fin de la chaîne de caractères à extraire du cookie. Le cookie débutant par une chaîne de 9 caractères moncookie (correspondant au nom), l’information qui nous intéresse (c’est-à-dire le compteur) est située à partir du dixième caractère. Vous affectez, alors, le résultat de la recherche dans la chaîne de caractères à une variable (valeurcookie). Ensuite, il faut effectuer une conversion de la variable, car celle-ci est en format texte, or comme elle va être utilisée dans une incrémentation, il faut la convertir en numérique, grâce à la méthode parseInt(). La variable cookienum peut, alors, être incrémentée. Ensuite, il suffit d’afficher le résultat dans une boîte de dialogue et de réécrire le cookie avec la nouvelle valeur. Dans le cas où il n’existe pas de cookie présent, il faut le créer et lui affecter la valeur 1 (puisque c’est la première visite).

10. Suppression d’un cookie

Pour supprimer un cookie, il suffit de le rappeler en lui réaffectant une date d’expiration antérieure à la date du jour. Une fois le navigateur fermé, le cookie sera supprimé.

Exemple : supprimer le cookie moncookie et afficher une boîte de dialogue pour informer le visiteur de la suppression.




```
<script type="text/javascript" language="JavaScript">
if (document.cookie.length>0) {
alert("Mon cookie est déjà présent");
document.cookie="moncookie=mavaleur;expires=Thu,01-Jan-1980 00:00:01
GMT;domain=www.mondomaine.com;path=/";
alert("Le cookie moncookie a été supprimé");
}
else {
alert("Le cookie moncookie n’est pas présent");
}
</script>
```

Comme d’habitude, il faut tester la présence du cookie par l’intermédiaire de document.cookie.length. Si le résultat est supérieur à zéro, c’est que le cookie est présent. Vous affichez alors, une boîte de dialogue. Ensuite, il suffit de créer un cookie portant le même nom mais avec une date d’expiration dépassée (ici, le premier janvier 1980). Cela suffit à supprimer le cookie indésirable. Si le cookie n’est pas présent, vous affichez simplement une boîte de dialogue informant le visiteur.

JavaScript et CSS

HTML, JavaScript et CSS peuvent interagir dans l'élaboration d'une page web. En effet, le langage HTML ne permet pas un contrôle approfondi des éléments composant une page. Ainsi, les pages écrites uniquement avec HTML présentent peu d'originalité. De nombreuses pages utilisent, de nos jours, des procédés techniques différents et variés, permettant d'ajouter des animations, des sons, voire de petites vidéos. Les fichiers d'animations au format flash en sont un exemple. Le DHTML *(Dynamic HyperText Markup Language)* est une alternative à Flash. Il combine HTML, CSS, le modèle d'objet DOM *(Document Object Model)* et JavaScript pour donner aux pages web un aspect graphique amélioré et un début d'interactivité (l'interactivité totale ne pouvant se concevoir qu'avec des langages de type serveur tel PhP ou Asp). Le DHTML n'est donc pas un langage de programmation à part entière mais une combinaison de ces trois techniques. Dans cette combinaison, JavaScript tient une place centrale. En effet, avec JavaScript, il est possible de manipuler le DOM et ses objets. Malheureusement le DOM n'a pas été implémenté de la même manière par tous les éditeurs des navigateurs. Aussi, il est parfois nécessaire de rédiger, au préalable, un script de détection de navigateurs et d'écrire, par la suite, autant de scripts qu'il y a de versions de navigateurs !!! Ces différences impliquent donc une bonne connaissance du DOM et de ses méthodes d'accès aux objets pour rédiger des scripts DHTML.

1. Le DOM (Document Object Model)

Le DOM est, en fait, une description hiérarchique des éléments composants une page web. C'est, donc, grâce à cette structure hiérarchisée que les langages de programmation (dont le JavaScript), peuvent accéder aux objets présents dans la page. Avec le DOM, le programmeur dispose d'un accès total à la page, lui permettant d'en modifier l'apparence mais aussi le contenu. Le DOM est une API (*Application Programming Interface* pour interface de programmation), totalement indépendante de la plate-forme et du langage qui la manipule. C'est le W3C *(World Wide Web Consortium)* qui a défini les différentes versions du modèle DOM depuis 1998. Il est possible de trouver une description complète du DOM, directement sur le site Internet du W3C, à l'adresse suivante : http://www.w3.org/DOM/ en version anglaise.

La première version, baptisée DOM1, est sortie en 1998 et a défini la représentation d'un document html ou xml *(eXtensible Markup Language)*, sous la forme d'un arbre. La notion de nœud a été retenue pour donner cette représentation hiérarchique de la page web. La seconde version, baptisée DOM2, est sortie en mars 2000 et a ajouté quelques fonctions permettant d'identifier les éléments d'une page. À la date de rédaction de cet ouvrage (en mai 2008), le W3C travaille à la sortie de la version 3 du DOM.

Une des critiques formulées au sujet du modèle DOM est qu'il lui est nécessaire de charger en mémoire, l'ensemble des objets d'une page web pour pouvoir fonctionner, ce qui peut rendre le temps de traitement excessivement long. Le DOM est une API fondée sur les arbres alors qu'il existe une autre méthode, baptisée SAX, et qui permet de remédier à cet épineux problème, car fondée sur les évènements. Pour notre part, nous nous limiterons à quelques techniques de base fondées sur le DOM et JavaScript pour interagir dans la page web.

2. Notions essentielles du DOM

Avec le DOM, le programmeur dispose d'objets, de propriétés et de méthodes lui permettant de manipuler tous les composants de la page, tout comme avec JavaScript. Cependant, avec le JavaScript, il n'est pas possible de manipuler certains objets (le contenu des cellules d'un tableau, par exemple), alors qu'en combinant JavaScript et DOM tout est faisable (ou presque). Une des notions essentielles du DOM est celle de nœud, qui correspond à un élément spécifié dans le code HTML d'une page web. Il existe trois types de nœud : les nœuds éléments, les nœuds attributs et les nœuds texte. Les nœuds sont, eux-mêmes, hiérarchisés.

Afin de mieux identifier la structure DOM composant une page HTML, il est possible d'utiliser l'outil DOM inspector inclus dans la version 2.0.0.14 de Firefox/Mozilla. Cet outil permet de représenter visuellement les éléments du DOM composant la page. Pour y accéder, il faut passer par le menu **Outils**, puis choisir Inspecteur DOM ou encore utiliser le raccourci-clavier [Ctrl] [Shift] **I**. Pour Internet Explorer, il existe également un certain nombre d'outils parmi lesquels nous pouvons citer DebugBar disponible gratuitement (sous réserve d'une utilisation personnelle), en téléchargement, à l'adresse : http://www.debugbar.com/

Après l'installation, l'outil est disponible sous la forme d'une barre d'outils à activer par le menu **Affichage** et l'option **Barre d'outils/ Debugbar**.

Comme indiqué précédemment, les dernières versions des navigateurs sont compatibles avec le DOM. Cependant, si le visiteur ne dispose pas d'une version récente, il lui sera difficile d'accéder aux fonctionnalités qui ont été développées. Il est donc préférable, d'effectuer, tout de même, un test de compatibilité. Par la suite, il sera possible soit d'afficher un message d'avertissement, soit d'orienter le script vers une partie prenant en charge les particularités de son navigateur. Pour y parvenir, il suffit de contrôler si celui-ci supporte bien l'accès aux objets par l'intermédiaire de la méthode getElementBy() de l'objet document. La syntaxe à utiliser est donc la suivante :

```
document.getElementById ;
```

Exemple : déterminer si le navigateur utilisé est compatible avec le DOM.




```
<script language="javascript">
if (document.getElementById) {
alert("Votre navigateur est compatible");
}
else {
alert("Dommage ! Votre navigateur n’est pas compatible")
}
</script>
```

En règle générale, ce script permet de déterminer les versions de navigateur supérieures à la version 4 d’Internet Explorer et Netscape. Il existe, cependant, d’autres paramètres qu’il est possible d’utiliser pour affiner la détection. Ainsi, en utilisant la syntaxe `document.all&&getElementById`, il est possible de savoir si le navigateur est une version 5 ou plus d’Internet Explorer. De la même manière, pour savoir si la version de Netscape est au moins égale à la version 6, il faut utiliser `document.all&&!getElementById`.

D’un point de vue syntaxique, vous utilisez les méthodes du DOM à l’intérieur du script JavaScript pour accéder aux éléments.

3. Les méthodes DOM pour accéder aux éléments

Avec le DOM, il existe des méthodes qui permettent d’accéder directement aux éléments d’une page, sans passer par la méthode de syntaxe pointée classique window.document.form…

Quatres méthodes principales permettent une manipulation des éléments composant une page :

Méthodes	**Résultat**
`getElementById()`	Sélectionne un élément en fonction de son identifiant.
`getElementsByName()`	Sélectionne un ou plusieurs éléments en fonction d’un nom passé en argument.
`getElementsByTagName()`	Sélectionne un ou plusieurs éléments en fonction d’un nom de balise passé en argument.
`innerHTML()`	Permet de lire ou d’assigner une valeur à un élément.

La première méthode getElementById() nécessite l’utilisation de la valeur de la balise id pour retrouver l’élément dans la page. Sa syntaxe est la suivante :

```
document.getElementById(’element a manipuler’)
```

Exemple : accéder à la valeur contenue dans un élément, par l’intermédiaire des méthodes proposées par le DOM.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Acces au DOM</title>
<script language="javascript">
function acces(){
document.getElementById('element1').innerHTML="Texte modifié par
DHTML";
}
</script>
</head>
<body>
<div id="element1">Texte de l'élément 1 </div>
<script language="javascript">
alert(document.getElementById('element1').innerHTML);
</script>
<input type="button" name="Submit" value="Modifier"
onClick="acces()">
</body>
</html>
```

Le script s'exécute au moment du clic sur le bouton **Modifier**. Auparavant, un script, placé dans le corps de la page, accède à la valeur de l'élément identifié par l'id 'element1' pour en afficher sa valeur. La fonction access permet d'accéder à l'élément pour en modifier sa valeur, toujours par l'intermédiaire de la méthode innerHTML(). Ceci démontre bien la possibilité de lecture et d'écriture de cette méthode.

Exemple : afficher le nombre des éléments ainsi que leur nom présents dans une page.




```
<script language="javascript">
var nombre=document.all.length;
alert("le nombre d'éléments présents dans le code de cette page est
de :"+nombre);
for (i=0; i<nombre;i++) {
nomelement=document.all(i).tagName;
afficheelement="Le nom de l'élément "+i+" est "+nomelement;
alert(afficheelement);
}
</script>
```

Le script commence par compter le nombre d'éléments du DOM, qui composent la page, puis l'affiche dans une boîte de dialogue. Ensuite, le script passe en revue l'ensemble des éléments identifiés par leur numéro d'indice et récupère leur nom en utilisant l'attribut tagName.

Le fait de pouvoir accéder aux éléments de la page par l'intermédiaire du DOM ouvre de nouvelles possibilités, notamment par l'association de JavaScript et des feuilles de styles en cascades qui permettent d'améliorer la mise en forme des pages Web.

4. Mise en forme de page web grâce à JavaScript et CSS

Quelques notions de bases sont indispensables à une bonne utilisation des CSS.

5. Premières notions de CSS

L'utilisation des CSS *(Cascading StyleSheets)* ou feuilles de style en cascade est un concept qui consiste à définir des propriétés de mise en forme et de les appliquer ensuite à tout ou partie d'un document. Le procédé présente de nombreux avantages et permet d'obtenir une présentation homogène, facilement modifiable ultérieurement. Une fois les feuilles de styles définies et appliquées aux pages d'un site, il est très facile d'en modifier l'apparence uniformément. En effet, la modification d'un style sera automatiquement répercutée sur l'ensemble des pages l'utilisant. Les feuilles de styles sont aujourd'hui compatibles avec toutes les dernières versions des navigateurs (version 4 d'Internet Explorer et Netscape Navigator, version 5 d'Opéra).

Afin de définir exactement cette compatibilité avec les standards définis par le W3C, il est important d'ajouter au code HTML de chaque page, une ligne d'instructions, appelée DTD *(Document Type Definition)*, qui clarifie les modalités de prise en charge des CSS. Sans entrer trop dans le détail, il suffit de savoir qu'il en existe trois types, du plus strict au plus large.

Le docType le plus commun et qui facilitera l'apprentissage est le suivant :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

En plus de cette définition de docType, il est souhaitable d'ajouter une balise Meta indiquant le type de CSS au navigateur et/ou aux moteurs de recherche.

```
<meta http-equiv="Content-Type" content="text/css">
```

Tout comme le JavaScript, le code CSS, définissant les caractéristiques du style, peut s'intégrer directement dans la page HTML entre les balises <HEAD> et </HEAD> ou dans un fichier texte externe, enregistré au format CSS. Nous ne traiterons pas ici de l'insertion de fichier CSS, car pour une meilleure lisibilité, il est préférable de disposer de l'ensemble du code dans la même page. Ainsi, pour insérer le code CSS, il est nécessaire (tout comme pour JavaScript), de l'inclure à l'intérieur des balises <style css> et </style>.

L'application de styles CSS se fait par l'intermédiaire de règles, composées d'un sélecteur de balise et d'une déclaration. Le sélecteur de balises correspond à la balise à laquelle le style devra être appliqué. La déclaration, quant à elle, comprend des propriétés auxquelles sont associées des valeurs. La composition d'une règle CSS se présente donc ainsi :

```
sélecteur {Propriété : valeur}
```

Où sélecteur correspond à une balise HTML, Propriété à une propriété de la balise et valeur à une valeur affectée à la propriété.

En somme, il faut définir sur quoi vous voulez appliquer le style et de quoi se compose le style. Pour prendre un exemple concret, définissons un style qui permette d'écrire avec la police Tahoma en gras et en rouge. Il est prévu que ce style s'applique à un texte compris dans la balise h1 (En-tête1).

En partant du code suivant de la page HTML :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Mon premier style CSS</title>
</head>
<body>
<h1>Texte en Tahoma, rouge et gras</h1>
</body>
</html>
```



Il faut ajouter la définition du style et son application dans le code de la page, qui devient :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Mon premier style CSS</title>
<style css>
h1{font-family: Tahoma;font-style: bold;color: red}
</style>
</head>
<body>
<h1>Texte en Tahoma, rouge et gras</h1>
</body>
</html>
```

Il n'est pas utile d'ajouter du code au niveau des balises h1, le simple fait d'indiquer la balise dans la définition de la règle, permet d'appliquer automatiquement le style CSS.

Le code CSS peut se trouver n'importe où dans la page, et même en dessous de la balise h1, en fin de partie body.

Il est possible d'appliquer le même style à plusieurs éléments, il suffit de les séparer par une virgule.

Une notion fondamentale de CSS réside dans le fait que les styles peuvent se cumuler (d'où leur nom). Ainsi, en reprenant l'exemple précédent et en créant une nouvelle règle permettant de mettre le texte en italique bâti sur le sélecteur h1, le texte prend tous les formatages décrits.

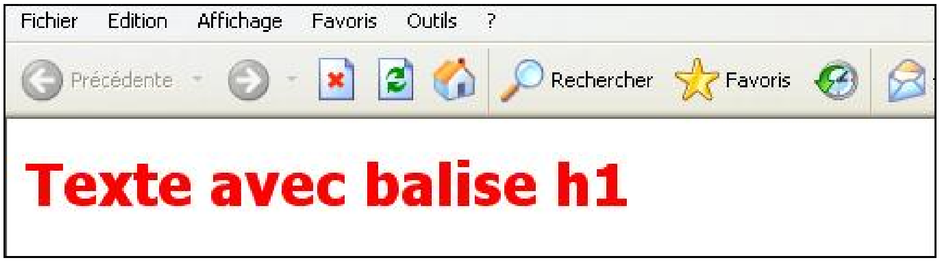
```
<style css>
h1{font-family: Tahoma;font-style: bold;color: red}
h1{font-style:italic}
</style>
```

L'ensemble des règles de style CSS peut être inclus à l'intérieur d'une classe qui sera appelée au moment de l'utilisation de la balise HTML. Il suffit d'ajouter un point, avant de définir l'ensemble des propriétés du style en accolades.

Pour appliquer le style à la balise, il faut utiliser l'instruction :

```
<h1 class="changerstyle">
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Mon premier style CSS</title>
</script>
<style css>
changerstyle {
font-family: Tahoma;font-style: bold;color: red; font:italic;
}
</style>
</head>
<body>
<h1 class="changerstyle">Texte avec balise h1</h1>
</body>
</html>
```



Dans ce script, la première étape consiste à déclarer la classe changerstyle et à définir les valeurs des propriétés. Ensuite, il suffit d'appeler la classe changerstyle à l'intérieur de la balise sur laquelle elle sera appliquée.

La liste des propriétés disponibles est importante, ce qui démontre l'étendue des possibilités offertes par l'utilisation des CSS.

Propriétés des polices	
Propriété	**Rôle**
font-family	Correspond à un ou plusieurs nom(s) de polices ou de familles de polices. Si plusieurs polices sont définies, la première trouvée sur le système de l'utilisateur sera utilisée.
font-style	Correspond au style d'écriture.
font-weight	Correspond à la graisse (épaisseur) de la police.
font-size	Correspond à la taille de la police.
font-variant	Correspond à une variante (petites majuscules).
font	Raccourci permettant de mettre toutes les propriétés.

Propriétés des paragraphes	
Propriété	**Rôle**
color	Correspond à la couleur du texte.
line-height	Correspond à l'interligne.
text-align	Correspond à l'alignement du texte.
text-indent	Correspond à l'indentation (retrait du texte).
text-decoration	Correspond à une décoration du texte.
text-shadow	Correspond à l'ombrage texte, respectivement décalage à droite, en bas, rayon de l'effet de flou et couleur.
text-transform	Correspond à la casse du texte.
white-space	Correspond à une césure.
word-spacing	Correspond à l'espacement des mots.
width	Correspond à la longueur d'un élément de texte ou d'une image.

height	Correspond à la hauteur d'un élément de texte ou d'une image.

Propriétés des couleurs et arrière-plan	
Propriété	**Rôle**
background-color	Correspond à la couleur d'arrière plan.
background-image	Correspond à l'image d'arrière-plan.
background-repeat	Correspond à la façon de répéter l'arrière-plan.
background-attachment	Indique si l'image reste fixe avec les déplacements de l'écran.
background-position	Correspond à la position de l'image par rapport au coin supérieur gauche.
background	Raccourci pour les propriétés d'arrière-plan.

Propriétés des marges	
Propriété	**Rôle**
margin-top	Correspond à la valeur de la marge supérieure.
margin-right	Correspond à la valeur de la marge de droite.
margin-bottom	Correspond à la valeur de la marge inférieure.
margin-left	Correspond à la valeur de la marge de gauche.
margin	Correspond à la valeur de la marge de gauche.

Propriétés des bordures	
Propriété	**Rôle**
border[-top -left -bottom -right]-width	Correspond à l'épaisseur de la bordure [supérieure, de gauche, inférieure ou de droite].
border[-top -left -bottom -right]-color	Correspond la couleur de la bordure [supérieure, de gauche, inférieure ou de droite].
border[-top -left -bottom -right]-style	Correspond au style de la bordure [supérieure, de gauche, inférieure ou de droite].
border-collapse	Correspond à la fusion de bordures.
border	Raccourci global les propriétés de bordure.

Propriétés des espaces intérieurs	
Propriété	**Rôle**
padding-top	Correspond à l'espace intérieur entre l'élément et la bordure supérieure.

padding-right	Correspond à l'espace intérieur entre l'élément et la bordure droite.
padding-bottom	Correspond à l'espace intérieur entre l'élément et la bordure inférieure.
padding-left	Correspond à l'espace intérieur entre l'élément et la bordure gauche.
padding	Raccourci vers l'ensemble des propriétés d'espace intérieur.

Propriétés des tableaux	
Propriété	**Rôle**
border-collapse	Correspond à la fusion des bordures des cellules (collapse) ou non (separate).
border-spacing	Correspond à l'espacement des cellules.
caption-side	Correspond au positionnement de la légende du tableau.
empty-cells	Correspond à l'affichage (show) ou au masquage (collapse) des cellules vides.
table-layout	Définit une largeur fixe ou variable.
speak-headers	Propriété pour sourds et malentendants, indiquant le comportement lors de la lecture des cellules d'en-tête d'un tableau.

Propriétés des listes	
Propriété	**Rôle**
list-style-type	Correspond au type de puces et de numérotation.
list-style-image	Permet de personnaliser les puces avec une image.
list-style-position	Correspond au retrait des puces.
list-style	Raccourci vers les propriétés de liste.

Propriétés de mise en page	
Propriété	**Rôle**
@page	Définit la mise en page de l'impression.
size	Correspond au format de l'impression.
margin-top	Correspond à la marge supérieure.
margin-right	Correspond à la marge de droite.
margin-bottom	Correspond à la marge inférieure.
margin-left	Correspond à la marge de gauche.

marks	Traits de coupe et repères de montage.
page-break-before	Force le saut de page avant un élément.
page-break-after	Force le saut de page après un élément.
orphans	Évite les lignes orphelines en fin de page. Définit le nombre de ligne(s) minima à partir duquel un renvoi en page suivante est effectué.
widows	Évite les lignes veuves en début de page. Définit le nombre de ligne(s) minima à partir duquel un renvoi en page précédente est effectué.

L'ensemble de ces propriétés démontre bien le vaste champ d'application des CSS. L'utilisation conjointe avec JavaScript permet de contrôler les différentes mises en forme, en fonction de situations ou de conditions particulières.

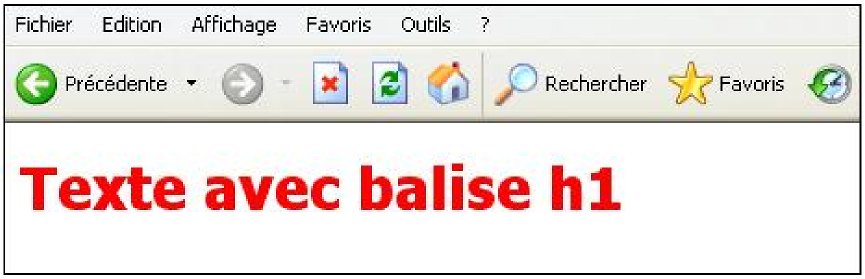
Interaction JavaScript/CSS

À présent que les règles CSS et la propriété className ont été abordées, il serait intéressant de pouvoir les associer à JavaScript pour améliorer l'affichage de certains éléments d'une page. Pour cela, il s'agit de commander l'application des feuilles de styles par des évènements.

1. Appliquer les feuilles de style grâce aux évènements

Le JavaScript permet de contrôler l'application des règles CSS définies. Il suffit d'ajouter un comportement pour exécuter une fonction appliquant le style CSS désiré.

Exemple : appliquer un style CSS sur un texte au moment du passage de la souris sur celui-ci (l'exemple reprend le code précédent).




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Mon premier style CSS</title>
<script language="javascript">
function changer(h1) {
h1.className="changerstyle";
}
</script>
<style css>
.changerstyle {
font-family: Tahoma;font-style: bold;color: red; font:italic;
}
</style>
</head>
<body>
<h1 class="Aucun style CSS" onMouseOver="changer(this)">Texte avec
balise h1</h1>
<p class="changerstyle"> </p>
</body>
</html
```

Au passage de la souris, la fonction changer s'exécute et prend l'objet this (ici la balise h1), comme argument. La fonction changer indique, ensuite, d'appliquer le style CSS nommé changerstyle.

2. Utiliser les structures de contrôles pour piloter l'application des styles CSS

Il est également possible d'utiliser les structures de contrôle fournies par JavaScript pour élaborer des scripts plus complexes, permettant des effets très utiles. Par exemple, l'usage des styles CSS peut simplifier la recherche d'informations dans un tableau où se trouve une grande quantité d'informations (même si pour l'exemple, il s'agit d'un tableau simplifié).

Exemple : modifier l'arrière-plan des cellules du tableau pour qu'il devienne rouge avec le texte blanc et gras, au passage de la souris.

A	B	C
1	2	3
4	5	6

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Javascript et CSS</title>
<style css>
.cellulesurbrillance {
color:#FFFFFF; font-weight:bold;
background-color:#CC0000;
}
</style>
<script language="javascript">
function allumecellule(cell) {
if (cell.className="Aucun style CSS") {
cell.className="cellulesurbrillance";
}
}
function eteincellule(cell) {
if(cell.className="cellulesurbrillance") {
cell.className="Aucun style CSS";
}
}
</script>

</head>
<body>
<table width="50%" border="1">
  <tr>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">A</div></td>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">B</div></td>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">C</div></td>
  </tr>
  <tr>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">1</div></td>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">2</div></td>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">3</div></td>
  </tr>
  <tr>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">4</div></td>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">5</div></td>
    <td class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)"><div align="center">6</div></td>
  </tr>
</table>
</body>
</html>
```

La première étape consiste à définir la classe et ses règles : `color:#FFFFFF` et `font-weight:bold` pour le texte blanc et gras et `background-color:#CC0000` pour le fond de la cellule. Ensuite, vous retrouvez le script JavaScript découpé en deux fonctions, une pour « allumer » la cellule (allumecellule) lors du passage de la souris, l'autre pour l'éteindre lorsque la souris la quitte (eteincellule). La première fonction teste si la classe est égale à « Aucun style CSS », qui est le style par défaut. Si c'est le cas, elle applique le style cellulesurbrillance. La seconde fonction fait le même travail mais avec une logique inverse, c'est-à-dire en testant si le style est cellulesurbrillance et dans ce cas, appliquer le style « Aucun style CSS ». Ensuite, il faut appliquer, pour chaque cellule du tableau, deux comportements, un premier pour déclencher la première fonction, un second pour déclencher la deuxième. Ainsi, en balayant le tableau, la cellule, en

dessous de laquelle la souris est positionnée, est facilement repérable.

Il est possible de modifier légèrement ce script pour mieux visualiser les lignes plutôt que les cellules. Dans ce cas, c'est la balise <tr>, qui correspond à la ligne complète du tableau, qui est à l'origine du changement de mise en forme à la place de la balise <td>, correspondant à la cellule.

```
<tr class="Aucun style CSS" onMouseOver="allumecellule(this)"
onMouseOut="eteincellule(this)">
```

3. Modifier l'affichage/masquage des éléments par JavaScript

Il est possible, également, d'accéder aux propriétés de style par JavaScript. Cela permet notamment d'afficher, de masquer des blocs de textes inclus dans des balises de type `<div>`.

Pour rappel, ce type de balise permet de constituer des blocs de texte et s'apparente à la balise `<span>`.

C'est la propriété `display` qui permet d'afficher ou de masquer les blocs. Il suffit de lui attribuer la valeur `none` pour cacher ou `block` pour afficher.

Il est possible d'utiliser une autre propriété à la place de display, il s'agit de la propriété `visibiliy` pouvant avoir comme valeur, `visible` ou `hidden` (caché). Dans ce cas, le résultat est légèrement différent puisque la place occupée par le bloc de texte n'est pas libérée.

Cet exemple est surtout utile sur des tableaux comprenants de nombreuses lignes et dans lesquels la place disponible n'est pas suffisante. Ici, par souci de simplification, le nombre de lignes est réduit à quatre.

Exemple : afficher/masquer des informations dans un tableau, au passage de la souris sur une cellule. Le message pourra être en rouge ou en vert selon les situations.

Date	Formation	Lieu
13 et 14 juin **Places disponibles**	JavaScript	Metz
22, 23 et 24 juin	HTML	Nancy
25 et 26 juin	JavaScript	Paris
29 et 30 juin	HTML	Strasbourg

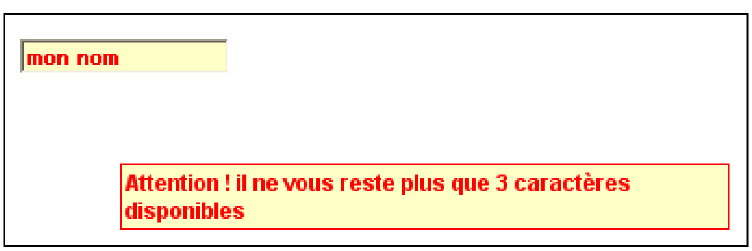
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Javascript et CSS</title>
<script language="javascript">
function efface(element) {
var tabelements=new Array();
tabelements=["1306","2206","2506","3006"];
for (i=0;i<tabelements.length;i++) {
document.getElementById(tabelements[i]).style.display='none';
}
}
function afficher(nom) {
document.getElementById(nom).style.display='block';
}
function masquer(nom) {
document.getElementById(nom).style.display='none';
}
</script>
<style type="text/css">
.Stylerouge {
    color: #FF0000;
    font-weight: bold;
}
.Stylevert {
    color: #006600;
    font-weight: bold;
}
```

```
</style>
</head>
<body onLoad="efface()">
<p> </p>
<table width="50%" border="1">
  <tr>
    <td ><span class="Style3">Date</span></td>
    <td ><span class="Style3">Formation</span></td>
    <td ><span class="Style3">Lieu</span></td>
  </tr>
  <tr>
    <td onMouseOver="afficher(1306)" onMouseOut="masquer(1306)">
13 et 14 juin
      <div class="Stylevert" id="1306">Places disponibles</div></td>
    <td onClick="masquer()">JavaScript</td>
    <td onClick="masquer()">Metz</td>
  </tr>
  <tr>
    <td onMouseOver="afficher(2206)" onMouseOut="masquer(2206)">22,
23 et 24 juin
    <div class="Stylerouge" id="2206">Plus de places
disponibles</div></td>
    <td onClick="masquer()">HTML</td>
    <td onClick="masquer()">Nancy</td>
  </tr>
  <tr>
    <td onMouseOver="afficher(2506)" onMouseOut="masquer(2506)">
25 et 26 juin
    <div class="Stylerouge" id="2506">Plus de places
disponibles</div></td>
    <td onClick="masquer()">JavaScript</td>
    <td onClick="masquer()">Paris</td>
  </tr>
  <tr>
    <td onMouseOver="afficher(3006)" onMouseOut="masquer(3006)">
29 et 30 juin
      <div class="Stylevert" id="3006">Places disponibles
</div></td>
    <td onClick="masquer()">HTML</td>
    <td onClick="masquer()">Strasbourg</td>
  </tr>
</table>
</body>
</html>
```

Le script porte sur quatre cellules d'un tableau et débute par la constitution d'un tableau permettant d'identifier les balises <div> en fonction de leur id. Cela permet, juste après, d'effectuer une boucle pour être certain que tous les blocs sont masqués. Le calcul du nombre d'éléments du tableau permet de déterminer le nombre d'itérations. Suivent, ensuite, les deux fonctions permettant l'affichage et le masquage, suivant le paramètre transmis au moment de l'évènement (onMouseOver et onMouseOut). Ainsi, au passage de la souris, l'évènement active la fonction afficher() et lui transmet l'Id du bloc à manipuler. Inversement, en quittant le texte, la fonction masquer() effectue le masquage du bloc concerné. Les deux styles CSS sont définis et appliqués aux balises <div>.

Voici un autre exemple d'utilisation un peu plus complexe permettant de gérer l'application de la visibilité.

Exemple : prévenir l'utilisateur, lors d'une saisie d'un champ de formulaire, du nombre de caractères disponibles dans le champ. Le champ lui-même doit prendre une présentation différente (fond jaune) pour prévenir de l'imminence de la limite.




```
<html>
<head>
<title>Avertissement limite champ de formulaire</title>
```

```
<script language="javascript">
function control(chaine) {
var longueur=0;
longueur=chaine.length;
limite=10-longueur;
var message="Attention ! il ne vous reste plus que "+limite+"
caractères disponibles";
if (limite<5 && limite>0) {
document.getElementById('alerte').style.visibility="visible";
document.getElementById('texte').className="fin";
document.getElementById('alerte').innerHTML=message;
}
else {
document.getElementById('alerte').style.visibility="hidden";
}
}
</script>
<style type="text/css">
.normal{
position:absolute;
left:64px;
top:83px;
height:25px;
width:329px;
visibility:hidden;
background-color:#FFFFCC;
border:1px solid double;
color: #FF0000;
padding:2px;
font-family:Arial, Helvetica, sans-serif;
font-size:12px;
font-weight:bold
}
.fin{
background-color:#FFFFCC;
color: #FF0000;
font-size:12px;
font-weight:bold}
</style>
</head>
<body>
<div id="alerte" class="normal"></div>
<form name="form1" method="post" action="">
  <p>
    <input name=texte type=text id="texte";" size=15 maxlength="15"
onKeyUp="control(this.value)">
  </p>
</form>
</body>
</html>
```

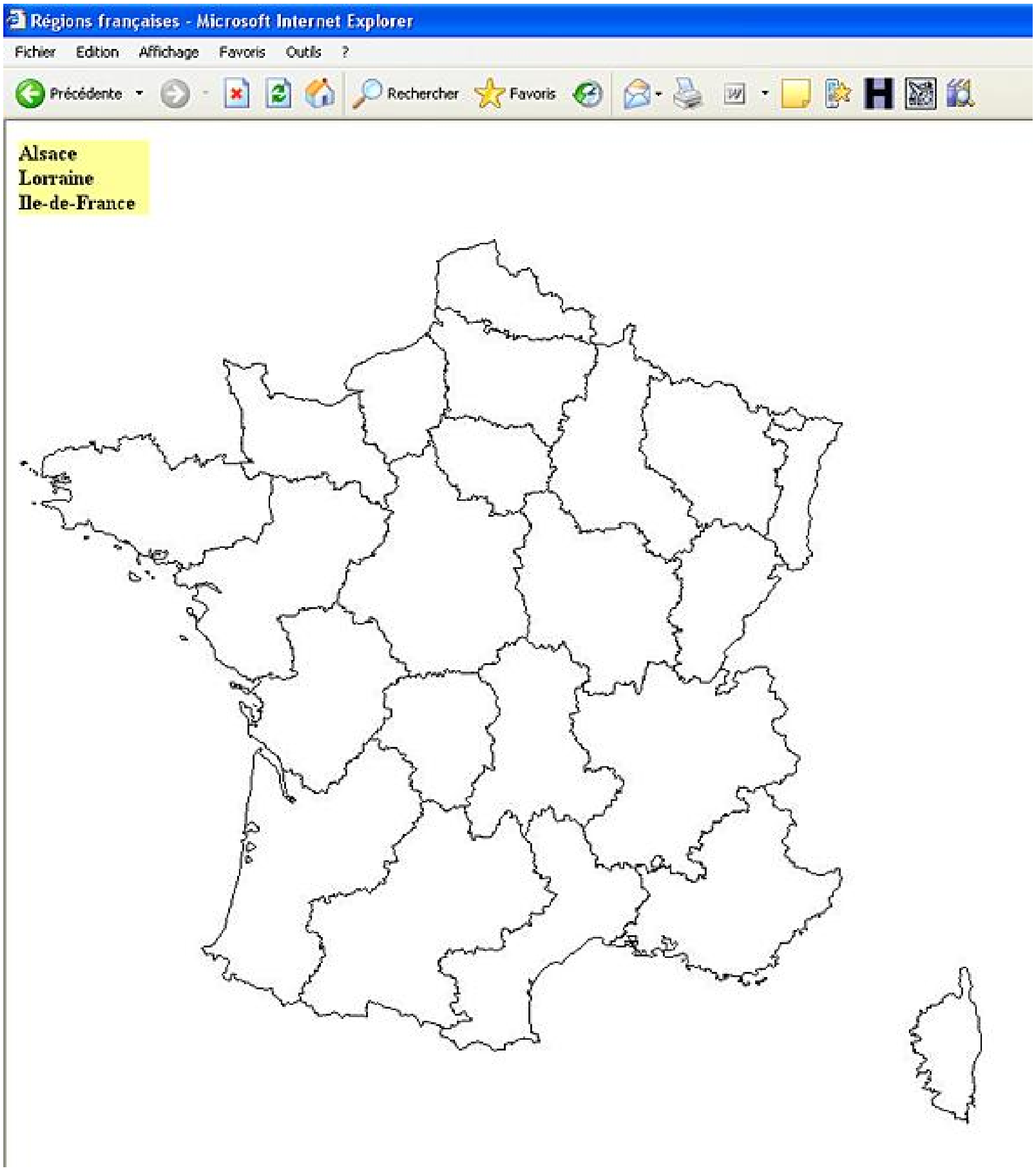
L'exécution de ce script dépend du nombre de caractères saisis dans le champ nommé texte. La fonction s'exécute, donc, à chaque relâchement de pression sur une touche. Le contenu du champ texte est, alors, renvoyé dans la fonction contrôle() comme argument. Il faut, ensuite, initialiser la variable longueur pour qu'elle prenne la nouvelle valeur de la longueur de la chaîne. Il est possible, ensuite, de calculer cette longueur par la propriété length de la chaîne de caractères. Étant donné que la limite de saisie du champ est fixée à 10 caractères, il est facile de déterminer par soustraction le nombre de caractères possibles restants. Juste après, le script construit une variable nommée message qui sera affichée dans le calque, un peu plus tard, par la méthode innerHTML et qui est composée de deux chaînes de caractères, séparées par la variable limite. C'est celle-ci qui permet, par la suite, de savoir si le nombre de caractères inclus dans le champ est compris entre 5 et 0. Si c'est le cas, il faut afficher le calque en modifiant sa visibilité puisque, par défaut, elle est fixée en hidden par l'intermédiaire d'un style CSS. Par la même occasion, il est possible de changer le style CSS du champ de formulaire pour être sûr d'attirer l'œil de l'utilisateur, en modifiant la couleur du fond pour la mettre en jaune. Enfin, il faut mettre à jour le contenu du calque par le message indiquant la fin conseillée de la saisie. Une fois la condition dépassée, le message s'efface pour laisser le champ en rouge.

4. Modifier la position des éléments par JavaScript

En plus de la visibilité, il est possible de modifier la position des éléments à l'aide de la propriété position. La position peut être définie soit de manière absolue (en pixels par rapport au coin supérieur gauche de l'objet), soit de manière

relative (par rapport à l'objet à l'origine de l'action). Pour le préciser, il convient d'ajouter la propriété position : relative ou position : absolute. Dans tous les cas, les propriétés left et top permettent de paramétrer cette position.

Exemple : créer une carte interactive permettant d'afficher des info-bulles avec le nom de la région française après un clic sur celle-ci. Par souci de simplification du code, le test prendra en charge seulement trois régions (l'Alsace, la Lorraine et l'Ile-de-France).




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Mon premier style CSS</title>
<script language="javascript">
function efface() {
document.getElementById("Alsace").style.visibility='hidden';
document.getElementById("Lorraine").style.visibility='hidden';
document.getElementById("IDF").style.visibility='hidden';
}
function coordonnees(event) {
var posx=event.clientX;
var posy=event.clientY;
if(posx>565&&posx<620&&posy>150&&posy<320) {
var posy=posy+25;
```

```
var posx=posx+25;
document.getElementById("Alsace").style.visibility='visible';
document.getElementById("Alsace").style.top=posy+"px";
document.getElementById("Alsace").style.left=posx+"px";
}
else if(posx>500&&posx<570&&posy>155&&posy<270){
var posy=posy+25;
var posx=posx+25;
document.getElementById("Lorraine").style.visibility='visible';
document.getElementById("Lorraine").style.top=posy+"px";
document.getElementById("Lorraine").style.left=posx+"px";
}
else if(posx>330&&posx<415&&posy>225&&posy<280){
var posy=posy+25;
var posx=posx+25;
document.getElementById("IDF").style.visibility='visible';
document.getElementById("IDF").style.top=posy+"px";
document.getElementById("IDF").style.left=posx+"px";
}
}
</script>
<style css>
.reponse{position:relative;background-color:#FFFF99;font-
weight:bold;width:100px;height:auto
}
</style>
</head>
<body>
<div class=reponse id="Alsace">Alsace</div>
<div class=reponse id="Lorraine">Lorraine</div>
<div class=reponse id="IDF">Ile-de-France</div>
<p><img src="../france_regions.jpg" width="740" height="680"
OnLoad="efface()" onClick="coordonnees(event)"></p>
</body>
</html>
```

Le script masque d'abord les calques éventuellement visibles au chargement de l'image par l'intermédiaire de la fonction efface(). Les autres fonctions s'exécutent au moment du clic sur une portion de l'image correspondant à la région. Pour déterminer la zone en pixels (à partir du coin supérieur gauche), il faut, d'abord, récupérer la position du curseur au moment du clic, puis effectuer une série de tests pour savoir si le curseur se trouve dans la zone définie. Pour définir la position du curseur, il est possible d'utiliser event.clientX et event.clientY. La définition de la zone correspondant à la région est faite de manière arbitraire, c'est-à-dire que le dessin des régions est réduit à un rectangle. Évidemment, pour ceux qui le désirent, cette zone peut être affinée. Il s'agit, ensuite, de redéfinir les coordonnées de la balise qui va comporter le nom de la région, en y appliquant un décalage de position de manière relative. Les paramètres d'affichage de la bulle étant définis dans le fichier CSS.

5. Modifier la taille des éléments par JavaScript

Il est possible, également, d'accéder aux propriétés de taille. Pour cela, il faut utiliser les propriétés de largeur (style.width) et de hauteur (style.height) :

Exemple : effectuer un changement de taille d'un calque après avoir demandé à l'utilisateur, les nouveaux paramètres.







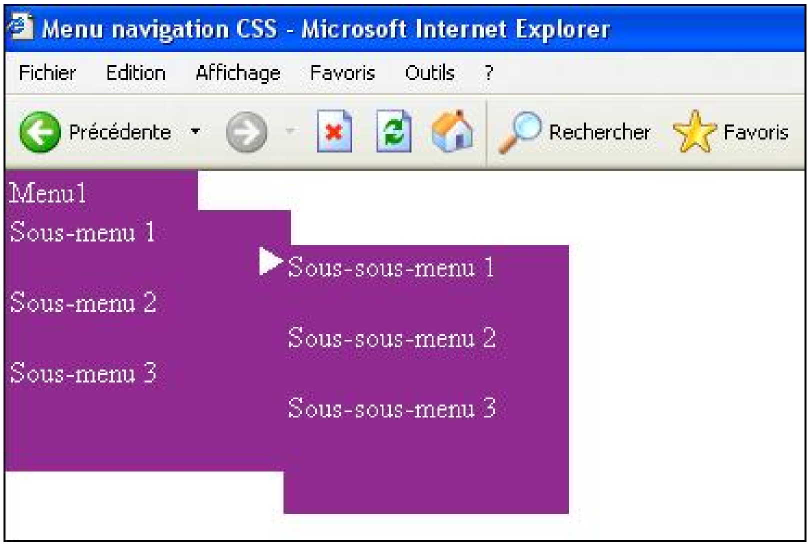

```
!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Modifier la taille</title>
<script language="javascript">
function changecalque(){
alert("Ce calque doit changer de taille");
var dimX=document.getElementById("calque1").offsetWidth;
var dimY=document.getElementById("calque1").offsetHeight;
alert("Les dimensions du calque sont : "+ dimX +" pixels en largeur
et "+dimY+" pixels en hauteur");
newdimX=prompt("Quelle est la nouvelle taille en largeur :",
"Saisissez ici la nouvelle taille");
newdimY=prompt("Quelle est la nouvelle taille en hauteur :",
"Saisissez ici la nouvelle taille");
newdimY=newdimY+"px";
newdimX=newdimX+"px";
document.getElementById("calque1").style.height=newdimY;
document.getElementById("calque1").style.width=newdimX;
}
</script>
<style type="text/css">
.calque1 {
position:absolute;left:300px;top:100px;color:#FF0000;
padding:10px;border:1px solid #000;background-color:#FFFF99;
}
</style>
</head>
<body>
<div class="calque1" id="calque1"
onClick="changecalque()">Calque1</div>
</body>
</html>
```

Ce script s'exécute au moment du clic sur le calque. La fonction changecalque() affiche tout d'abord une boîte de dialogue puis affecte aux variables dimX et dimY les valeurs en pixels de la taille du calque. Le script affiche ensuite ces valeurs dans une boîte de dialogue. Ensuite, il est demandé à l'utilisateur de saisir les nouvelles valeurs afin de changer les dimensions du calque. Ces valeurs sont stockées dans les variables nexdimX newdimY auxquelles il faut ajouter "px" pour obtenir une nouvelle chaîne correspondant par exemple à 100px. Enfin, il est possible de modifier les dimensions du calque en utilisant la méthode getElementById.

6. Modifier la superposition des éléments par JavaScript

Il est, également, possible de travailler avec la superposition d'éléments. La superposition est gérée par la propriété z-index. En affectant une valeur à z-index, il est possible de modifier l'ordre de superposition de l'objet.

Exemple : créer un menu de navigation en combinant les propriétés de visibilité, de position et de superposition.




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/css">
<title>Menu navigation CSS</title>
<script language="javascript">
function efface(numcalque) {
document.getElementById(numcalque).style.visibility='hidden';
document.getElementById('triangless1').style.visibility='hidden';
}
function affiche(calque) {
document.getElementById(calque).style.visibility='visible';
}
function masque(calque) {
document.getElementById(calque).style.visibility='hidden';
}
function affichetriangle(triangle) {
document.getElementById(triangle).style.visibility='visible';
document.getElementById(triangle).style.zIndex=2;
}
function recultriangle(triangle) {
document.getElementById(triangle).style.zIndex=0;
}
function effacetriangle(triangle) {
document.getElementById(triangle).style.visibility='hidden';
}
</script>
<style type="text/css">
.calque1{
    visibility:hidden;
    padding:2px;
    position:absolute;
    background:#993399;
    border:#000000:solid;
    left:0px;
    top:21px;
    height:137px;
    width:150px;
    color:#FFFFFF;
    background-color: #993399;
}
.menu1{
    position:absolute;
    padding:2px;
    background:#993399;
    border:#000000:solid;
    left:0px;
    top:0px;
```

```
    height:20px;
    width:100px;
    color:#FFFFFF}
.triangless1{
    visibility:hidden;
    padding:1px;
    position:absolute;
    padding:0px;
    left:136px;
    top:40px}
.calque2{
    visibility:hidden;
    padding:2px;
    position:absolute;
    background:#993399;
    border:#000000:solid;
    left:150px;
    top:40px;
    height:141px;
    width:150px;
    color:#FFFFFF}
.zoneface{
    position:absolute;
    background:#FFFFFF;
    border:#000000:solid;
    left:742px;
    top:161px;
    height:250px;
    width:350px;
    color:#FFFFFF}
</style>
</head>
<body>
<div class="menu1" id="Menu1" style="z-index:1"
onMouseOver="affiche('calque1')">Menu1</div>
<div class="calque1" id="calque1" style="z-index:1"
onMouseOver="affiche('calque1')">
  <div id="txtsous-menu1"
onMouseOver="affichetriangle('triangless1');affiche('ss1')"
onMouseOut="effacetriangle('triangless1'); masque('ss1')">
    <p>Sous-menu 1</p>
  </div>
  <div id="txtsous-menu2">
    <p>Sous-menu 2</p>
  </div>
  <div id="txtsous-menu3">
    <p>Sous-menu 3</p>
  </div>
</div>

<div class="calque2" id="ss1" style="z-index:1">
  <p>Sous-sous-menu 1</p>
  <p>Sous-sous-menu 2</p>
  <p> Sous-sous-menu 3</p>
</div>
<div class="zoneface" id="zoneface" style="z-index:0"
onMouseOver="masque('calque1'); masque('ss1')"></div>
<img src="triangle_mauve.jpg" width="15" height="19"
class="triangless1" id="triangless1" >
<img src="triangle_mauve.jpg" width="15" height="19"
class="triangless1" id="triangless2" >
</body>
</html>
```

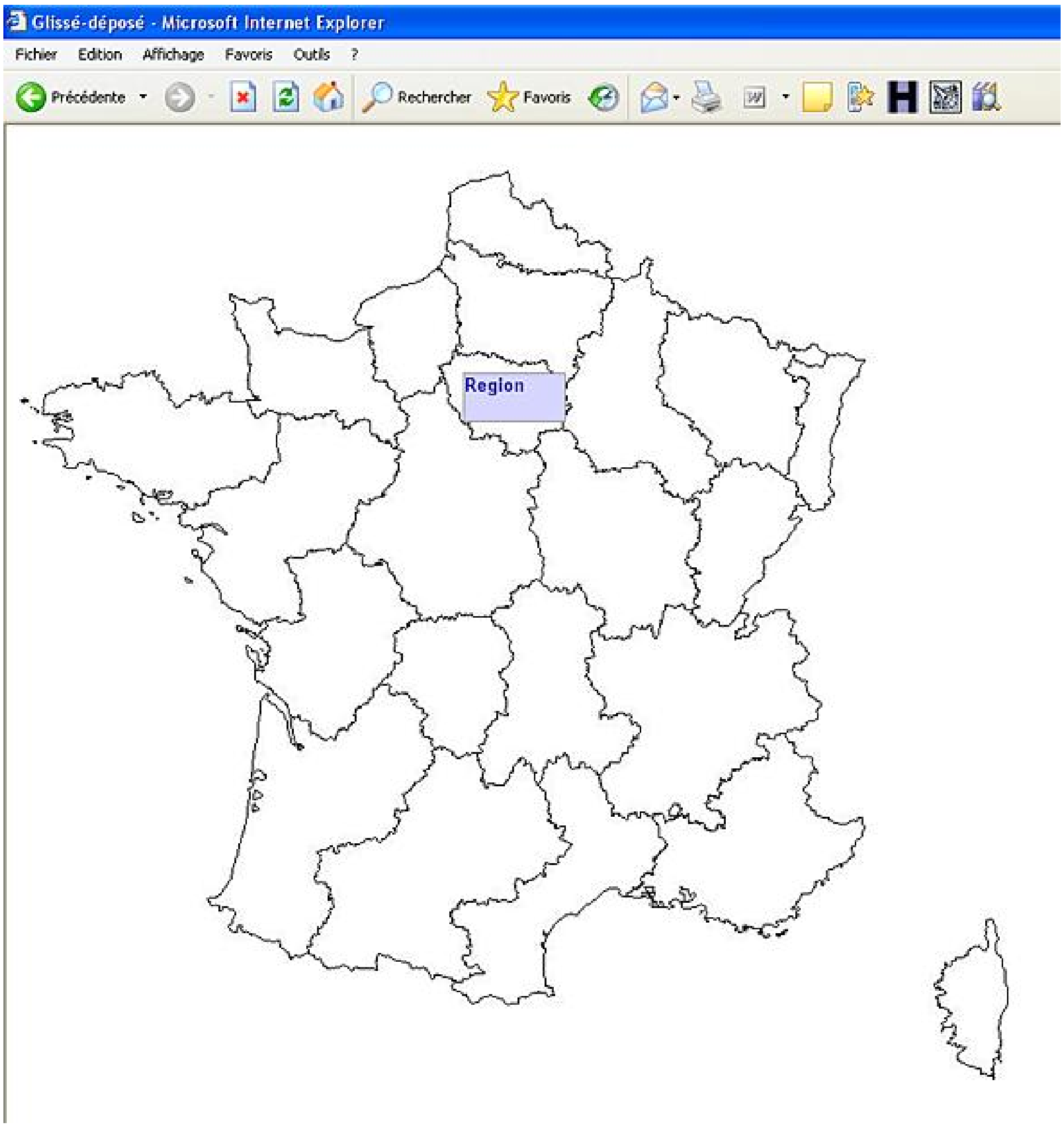
Le script se divise en plusieurs fonctions qui permettent d'afficher ou de masquer certains composants du menu (menu, sous-menus et triangle d'indication). Ce script se base sur l'affichage ou le masquage d'éléments, en fonction du survol sur un autre élément. Le principe de rapprochement/recul est, aussi, employé, notamment pour le triangle qui change d'indice de superposition, pour apparaître par-dessus les éléments de menu. La première fonction effacecalque() a pour rôle d'effacer les calques et le triangle d'indication par l'intermédiaire de la propriété de visibilité. Les autres fonctions permettent, à leur tour, d'afficher ou de masquer des éléments de menu, transmis en paramètre

au moment du survol. Ainsi, lorsque le premier texte du sous-menu est survolé, le triangle redevient visible et son indice z-index prend la valeur 2, ce qui lui permet de passer au-dessus des menus et sous-menus. À l'inverse, lorsque la souris quitte le survol du premier texte du sous-menu, le triangle prend une valeur de z-index égale à zéro et passe donc en-dessous du menu. Il est évident qu'il aurait été possible de masquer le triangle par la propriété de visibilité, cependant ce script est un bon exemple du mélange des propriétés de visibilité et de superposition.

7. Le drag-and-drop

Le drag-and-drop (glissé-déposé en français) permet de saisir un objet (généralement un calque), en cliquant dessus et de le déplacer, en maintenant enfoncé le bouton gauche de la souris. Une fois à la position désirée, il suffit de relâcher le bouton. Il n'existe pas de fonction prédéfinie de drag-and-drop dans JavaScript. Pour y arriver, il s'agit de trouver une astuce permettant de réaliser le déplacement.

Exemple : réaliser un script permettant de faire un glissé-déposé d'un calque, correspondant à l'étiquette d'une région française sur une carte de france.




```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<title>Glissé-déposé</title>
<script language="javascript">
var positionX=0, positionY=0;
var dimX=0, dimeY=0;
var mavar=1;
var nouvellepositionX=0, nouvellepositionY=0;
function depart(calque) {
dimX=positionX-document.getElementById(calque).offsetLeft;
```

```
dimY=positionY-document.getElementById(calque).offsetTop;
mavar=0;
}
function deplacement(page) {
if (document.all) {
positionX=event.clientX;
positionY=event.clientY;
}
else {
positionX=page.pageX;
positionY=page.pageY;
}
if(mavar!=1){
var nouvellepositionX=positionX-dimX;
var nouvellepositionY=positionY-dimY;
document.getElementById('calque1').style.left=
nouvellepositionX+"px";
document.getElementById('calque1').style.top=
nouvellepositionY+"px";
}
}
function arret(calque) {
calqueDragDrop="";
mavar=1;
}
document.onmousemove=deplacement;
document.onmouseup=arret;
</script>
<style type="text/css">
.calque {position:absolute;left:400px;top:100px;
cursor:move;width:75px;height:35px;font-size:10pt;
font-weight:bold;font-family:arial;border:1px solid #999
}
</style>
<body style="height:100%">
<div id="calque1" class="calque" style="top:25px; left:931px;
z-index:0; color:#009; background-color:#D5D5FF;"
onMouseDown="depart('calque1')">Region</div>
<p><br />
  <img src="france_regions.jpg" width="740" height="680">
</p>
</body>
</html>
```

Le script débute au chargement de la page et initialise à zéro l'ensemble des variables qui sont utilisées. La fonction depart() s'exécute au moment du clic, ce qui permet de récupérer les coordonnées de la souris par rapport au coin supérieur gauche du calque et de les stocker dans deux variables dimX et dimY. Par la même occasion, la variable mavar prend la valeur de 0 indiquant, ainsi, que le calque est sélectionné.

Au moment du déplacement de la souris, la fonction deplacement() teste le type de navigateur et permet de récupérer de deux manières différentes les coordonnées de la souris, suivant le type de celui-ci. Avec Internet Explorer, il s'agit de la méthode event.clientX() et event.clientY() alors qu'avec Mozilla/Firefox, il s'agit de la méthode pageX(), pageY(). En fonction de la position de la souris et si la variable mavar est différente de zéro (ce qui est normalement le cas puisque l'utilisateur a cliqué sur le calque), il s'agit de calculer la nouvelle position correspondante. Pour ce faire, il faut prendre la position de la souris et y soustraire le décalage du clic dans le calque (variable dimX et dimY). Le résultat est ensuite réaffecté au calque en passant par les méthodes document.getElementById('calque1').style.top() et document.getElementById('calque1').style.left().

Ajax et JavaScript

Tout d’abord, il convient de signaler, que cette partie de l’ouvrage ne prétend pas donner l’ensemble des connaissances en AJAX et ainsi, permettre de développer l’intégralité d’un site avec cette technologie. Il s’agit simplement d’en définir les contours et d’en connaître les principes de fonctionnement. Mais AJAX, en définitive, ça correspond à quoi ? Tout comme le DHTML, l’AJAX (*Asynchronous JavaScript And XML*) est un mélange de différentes techniques. Avec AJAX, il est possible d’effectuer des rechargements de données, en provenance du serveur, tout en restant sur la même page. En fait, un visiteur peut continuer à saisir des champs de formulaire alors qu’un contrôle est effectué en arrière-plan pour vérifier une saisie précédente avec les données présentes sur un serveur. Ce type de fonctionnement, appelé asynchrone, n’est pas le seul, car la technologie AJAX permet également d’exécuter des requêtes en mode synchrone (dans ce cas, la requête est exécutée sans qu’il soit nécessaire de recharger la page ou d’en charger une autre, mais la saisie simultanée est impossible, ce qui réduit son utilisation). En fait, les pages écrites avec de l’AJAX cumulent plusieurs technologies :

- Le code HTML qui reste au cœur de la page web, aidé par les feuilles de style CSS pour toute la partie de présentation des données.
- Le DOM pour l’accès aux éléments de la page, notamment par les méthodes getElementById ou getElementByName.
- Le code de programmation côté serveur, de type PHP ou ASP, qui permet de récupérer des informations du serveur comme l’objet XmlHttpRequest qui permet de contrôler l’interrogation des données du serveur.
- Les données en retour se présentent sous la forme d’un simple fichier texte ou JavaScript ou encore XML et doivent être traitées par JavaScript pour apparaître convenablement sur la page.

Comme vous le voyez, JavaScript se situe au cœur de ces différentes technologies et joue, un peu, le rôle d’un chef d’orchestre gérant l’ensemble des traitements. Par opposition à cette multitude de technologies, l’Ajax se fonde essentiellement autour d’un seul et même objet, nommé XmlHttpRequest.

1. L’objet XmlHttpRequest

Cet objet est identique pour tous les navigateurs récents, même si sa déclaration est différente entre Internet Explorer et les autres navigateurs. Cet objet est relativement ancien puisqu’il a été développé par Microsoft dès 1998 et implémenté dans la version 5 d’Internet Explorer. Les autres navigateurs ont, progressivement, pris en compte cet objet. Il est donc, important de connaître la compatibilité du navigateur.

Exemple : créer un script de vérification de compatibilité du navigateur et afficher le résultat dans une boîte de dialogue.




```
<script language="javascript">
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
alert("Votre navigateur est compatible pour AJAX");
}
else if(window.ActiveXObject){
xhr=new ActiveXObject("Microsoft.XMLHTTP");
alert("Votre navigateur est compatible pour AJAX");
}
else {alert("Votre navigateur n’est pas compatible avec AJAX");
}
</script>
```

Il s’agit, ici, de tenter de créer une nouvelle instance d’abord pour Firefox/Mozilla/Mozilla, puis, de manière différente, pour Internet Explorer (puisque basé sur un composant ActiveX), si la première n’est pas concluante.

Une fois la compatibilité déterminée, il reste à manipuler XmlHttpRequest, afin d’exécuter les interrogations de

données. XmlHttpRequest dispose de plusieurs méthodes et attributs à cet effet.

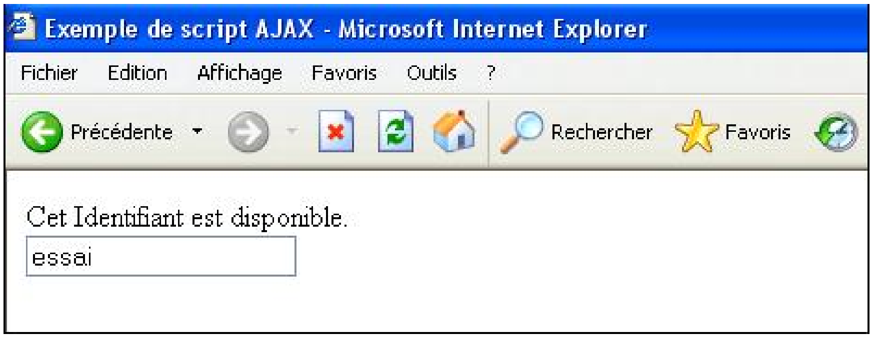
2. Les attributs de XmlHttpRequest

Attributs	Résultat
`onreadystatechange`	Affecte une fonction à chaque changement d'état dans le traitement de la requête en mode asynchrone.
`readyState`	Correspond à l'état de l'objet, tout au long du traitement de la requête. Elle comporte quatre valeurs : 0 pour non initialisée (Uninitialized), 1 pour ouvert (Open), 2 pour envoyée (Sent), 3 pour encours de réception (Receiving) et 4 pour prêt (Loaded).
`responseText`	Indique que la réponse devra être renvoyée sous forme texte.
`responseXML`	Indique que la réponse devra être renvoyée sous forme XML.
`status`	Correspond au code du statut de serveur (404 pour page non trouvée et 200 pour OK).
`statusText`	Correspond au message accompagnant le code statut.

Méthode	Résultat
`abort`	Abandonne la requête et réinitialise l'objet XmlHttpRequest.
`getAllResponseHeaders()`	Correspond aux en-têtes http de la réponse.
`getResponseHeader()`	Correspond à la valeur de l'en-tête indiquée en paramètre.
`open()`	Permet la connexion avec le serveur en passant les paramètres de méthode (GET, POST), d'URL et de type (synchrone ou asynchrone).
`send()`	Transmet une requête au serveur selon les méthodes GET ou POST.
`setRequestHeader()`	Affecte une valeur à une en-tête qui sera renvoyée lors de la requête.

L'ensemble de ces attributs et méthodes permet de mettre en œuvre des scripts écrits en JavaScript, pour accéder à des données présentes sur le serveur, sans qu'il soit nécessaire de recharger la page. C'est là toute la puissance d'Ajax ouvrant la voie de ce que l'on appelle, communément aujourd'hui, le web 2.0. Afin de bien observer les résultats des scripts suivants, il est important de travailler dans un environnement adéquat. Il est, par exemple, nécessaire de mettre sur serveur les fichiers comportant les données à récupérer.

Exemple : contrôler la saisie d'un champ pseudo pour renvoyer un message d'avertissement, si celui-ci a déjà été pris par un autre utilisateur.



Le fichier php :

```
<?php require_once('../../Connections/connex1.php'); ?>
<?php
mysql_select_db($database_connex1, $connex1);
$query_Recordset1 = ("SELECT Utilisateurs.Identifiant FROM Utilisateurs
WHERE Utilisateurs.Identifiant = '".$_GET["Identifiant"]."'");
$Recordset1 = mysql_query($query_Recordset1, $connex1) or
die(mysql_error());
$row_Recordset1 = mysql_fetch_row($Recordset1);
$totalRows_Recordset1 = mysql_num_rows($Recordset1);
//echo $Recordset1;
if ($row_Recordset1>0) {
echo 'déjapris';
}
?>
```

Le script PhP utilise le paramètre (Identifiant), renvoyé par le fichier html, pour extraire les données grâce à la requête. Dans le cas où la requête trouve des éléments répondants aux critères (c'est-à-dire si $row_Recordset1>0), cela signifie qu'il existe déjà un identifiant identique. Dans ce cas, le script renvoie la chaîne de caractères 'déjapris' comme réponse.

Le fichier avec script JavaScript :

```
<html>
<head>
<title>Exemple de script AJAX</title>
<script language="javascript">
var retour = "";
function verif(Identifiant){
 if (Identifiant.length >= 4) {
freponse(Identifiant);
}
}
function freponse(Identifiant) {
retour = connectURL('control.php?Identifiant=='+Identifiant);
if(retour=="déjapris"); {
document.getElementById('reponse').innerHTML = 'Cet Identifiant n\'est
pas disponible.';
}
document.getElementById('reponse').innerHTML = 'Cet Identifiant est
disponible.';
}
function connectURL(url)
{
if (window.XMLHttpRequest)
objXHR = new XMLHttpRequest();
else
{
if (window.ActiveXObject)
objXHR = new ActiveXObject("Microsoft.XMLHTTP");
alert(objXHR);
}
objXHR.open("GET",url,false);
objXHR.send(null);
if (objXHR.readyState == 4)
return objXHR.responseText;
```

```
else
return false;
}
</script>
</head>
<body>
<form>
<div id="reponse"></div>
<input type="text" name="Identifiant" onKeyUp="verif(this.value);" />
</form>
</body>
</html>
```

Dans ce script, c'est la fonction verif() qui lance le traitement à chaque relâchement d'une touche du clavier. Le contrôle de l'identifiant s'effectue seulement, si 4 caractères, au moins, ont été saisis. Dans ce cas, la fonction freponse() prend le relais. Elle définit une variable nommée retour correspondant au résultat retourné par le fichier PhP, par l'intermédiaire de la fonction connectURL() dont l'adresse du fichier et le critère d'interrogation sont passés en paramètres. La fonction connectURL() commence par tester le navigateur pour savoir s'il accepte l'objet XmlHttpRequest et construit l'objet objXHR. Il faut, ensuite, tester le statut de la connexion pour savoir s'il est possible d'envoyer une requête (objXHR.readyState == 4). Si c'est le cas, la fonction connectURL() retourne la réponse sous forme texte. Une fois la réponse retournée, la fonction freponse traite la valeur. En fonction du résultat du test, un message est envoyé au calque par l'intermédiaire de l'instruction innerHTML.

La puissance d'AJAX est ainsi démontrée et depuis quelques mois, le nombre de pages contenant du code AJAX croît de manière exponentielle. En fait, comme un effet de mode, développer en AJAX est devenu un must. Et comme toujours en de tel cas de figure, il est possible de retrouver AJAX même dans des pages qui n'en n'ont pas forcément besoin. Étant donné que le recours à AJAX multiplie les requêtes au serveur de données, il convient tout de même de limiter son utilisation à des cas bien précis.

Ce script démontre également que JavaScript est loin d'être un sous langage. Combinés avec des technologies récentes, les scripts rédigés en JavaScript peuvent être très élaborés et complexes à maintenir. Il existe, cependant, une alternative permettant de simplifier la rédaction de tels scripts. C'est le recours aux bibliothèques JavaScript qui permet, en outre, de découvrir de nouvelles utilisations.

Les bibliothèques JavaScript

Depuis le début de cet ouvrage, les principes de JavaScript ont été mis en application pour permettre d'ajouter plus de fonctionnalités aux pages HTML. Malgré tout, un tel développement nécessite un investissement en temps important, qui peut être source de découragement. Heureusement, il existe de nombreux framework (bibliothèques) JavaScript en téléchargement gratuit qui permettent non seulement un gain de temps considérable, mais offrent, aussi, de nombreuses possibilités supplémentaires. Il ne s'agit pas ici de toutes les passer en revue, mais simplement d'en montrer leur installation et leur principe de fonctionnement. Nous nous attarderons, plus particulièrement, sur la bibliothèque script.aculo.us qui permet de nombreux effets visuels et permet d'ajouter facilement beaucoup de fonctionnalités (autocomplétion, drag-and-drop...). Mais tout d'abord, il faut décrire la bibliothèque Prototype, à la base d'autres frameworks et qui est, donc, indispensable.

1. La bibliothèque Prototype

La bibliothèque Prototype est une bibliothèque permettant de simplifier la rédaction de scripts JavaScript. Les fonctionnalités proposées sont une sorte d'extension aux méthodes JavaScript. Cette bibliothèque fournit de nombreux raccourcis de codes intéressants et permet, également, de simplifier le traitement des requêtes AJAX.

Cette bibliothèque est disponible en téléchargement à l'adresse suivante : http://www.prototypejs.org/download

Cette bibliothèque est constituée d'un seul fichier qu'il est nécessaire d'appeler, lors de chaque usage, par la ligne suivante :

```
<script language="javascript" src="chemindufichier/prototype.js">
```

Il serait trop long de donner une description détaillée de la bibliothèque Prototype. En effet, elle sert de base à une autre bibliothèque script.aculo.us, fournissant de nombreuses fonctionnalités intéressantes (notamment au niveau graphique) et que nous détaillerons dans le paragraphe suivant.

2. La bibliothèque script.aculo.us

Cette bibliothèque est disponible en téléchargement gratuit à l'adresse suivante : http://script.aculo.us/downloads

La bibliothèque est constituée de plusieurs fichiers classés dans trois répertoires. Le répertoire lib contient la bibliothèque Prototype, nécessaire à l'utilisation des scripts de script.aculo.us. Le répertoire test contient deux pages permettant de visualiser et de tester les fonctionnalités de script.aculo.us. Ainsi, la page run_unit_tests vous permet de vérifier que les scripts s'exécutent bien sur le navigateur de test, il s'agit de laisser le test se dérouler et de contrôler le message de retour pour être sûr que l'ensemble fonctionne correctement. Enfin, le répertoire src comprend huit fichiers JavaScript correspondant au fichier de base scriptaculous.js ainsi que sept modules correspondant à des fonctionnalités précises :

Fichier	Fonctionnalités
effects.js	Effets spéciaux.
builder.js	Manipulation des objets HTML via DOM.
dragdrop.js	Effets de glisser-déposer.
sliders.js	Effets de glisser.
sound.js	Utilisation des sons.
control.js	Auto complétion.
unitest.js	Test des effets.

Afin de disposer des fonctionnalités de scriptaculo.us, il faut soit changer le fichier correspondant à l'effet désiré, soit changer le fichier scriptaculos.js après le fichier prototype.js. Les deux lignes suivantes doivent, donc, figurer dans les pages :

```
<script language="javascript" src="prototype.js" >
</script>
```

```
<script language="javascript" src="scriptaculous.js">
</script>
```

Le fichier scriptaculous.js fait ensuite appel aux différents fichiers (effects.js, sliders.js), en fonction des besoins du script de la page. Cela signifie que tous les fichiers sont chargés, avant même le moindre traitement. Cet aspect peut paraître contraignant pour les utilisateurs ne disposant pas d'un débit suffisant (même si cela devient de plus en plus rare). Afin d'alléger le temps de téléchargement, il est possible de faire appel au seul fichier concerné. Il est, par exemple, inutile de charger le fichier dragdrop si votre page ne comporte que des effets d'animation. Dans ce cas, il faut faire appel au seul fichier concerné, par la syntaxe suivante :

```
<script language="javascript"
src="chemindufichier/scriptaculous.js?load=effects"></script>
```

Les fichiers js doivent être placés sur le serveur, bien entendu. Grâce à cette mise en œuvre, il est possible d'accéder à de nouvelles méthodes et à de nouveaux mots clés.

Afin de simplifier la présentation, les scripts suivants font appel à la bibliothèque entière.

3. Les effets visuels avec script.aculo.us

Plutôt que de lister les effets visuels, le mieux est peut-être d'élaborer un script permettant de les appliquer.

Pour utiliser ces effets, il suffit de respecter la syntaxe suivante :

```
New Effect.nomdeleffet("nomelement",options)
```

Où `nomdeleffet` correspond à l'effet à utiliser parmi la liste des effets disponibles, `nomelement` correspond à l'élément sur lequel appliquer l'effet (généralement un calque) et `options` correspond au paramétrage de l'effet (position, dimension, etc.).

Exemple : créer une page permettant d'appliquer les différents effets sur un bloc de texte.




```
<html>
  <head>
    <title>Test effets avec scriptaculos</title>
    <script type="text/javascript" src="prototype.js"></script>
    <script type="text/javascript" src="/scriptaculous.js"></script>
      <script language="javascript">
      var el="element1";
      function souligne() {
      new Effect.Highlight(el);
      }
      function deplacer() {
      new Effect.MoveBy(el,100,100);
      }
      function echelle() {
      new Effect.Scale(el,150);
      }
      function bouger(){
      new Effect.Shake(el);
      }
      function disparaitre() {
      new Effect.Fade(el);
      }
      function apparaitre() {
      new Effect.Appear(el);
      }
      function deroule() {
      new Effect.BlindDown(el);
      }
      function enroule() {
      new Effect.BlindUp(el);
      }
      function fuit() {
      new Effect.DropOut(el);
```

```
      }
      function reduit() {
      new Effect.Fold(el);
      }
      function grossis() {
      new Effect.Grow(el);
      }
      function battre() {
      new Effect.Pulsate(el);
      }
      function nuage() {
      new Effect.Puff(el);
      }
      function rebouger() {
      new Effect.Shrink(el);
      }
      function retour() {
      new Effect.Squish(el);
      }
      function interrupt() {
      new Effect.SwichOff(el);
      }

      </script>
    <style type="text/css">
.Style1 {
left:800;
top:200;
      color: #FF0000;
      font-weight: bold;
}
    </style>
</head>
  <body>
      <div class="Style1" id="element1">
        <div align="center">Texte</div>
      </div>
      <div align="center">
        <table width="72%" border="0">

          <tr>
            <td colspan="15"><div align="center">Effets visuels de
script.aculo.us </div></td>
          </tr>
          <tr>
            <td width="5%"><input type="button" name="Submit15"
value="Echelle" onClick="echelle()"></td>
            <td width="5%"><input type="button" name="Submit13"
value="Souligner" onClick="souligne()"></td>
            <td width="5%"><input type="button" name="Submit14"
value="Deplacer" onClick="deplacer()"></td>
            <td width="5%"><input type="button" name="Submit"
value="Bouger" onClick="bouger()"></td>
            <td width="7%"><input type="button" name="Submit2"
value="Disparaitre" onClick="disparaitre()"></td>
            <td width="7%"><input type="button" name="Submit3"
value="Apparaitre" onClick="apparaitre()"></td>
            <td width="6%"><input type="button" name="Submit4"
value="Enrouler" onClick="enroule()"></td>
            <td width="7%"><input type="submit" name="Submit5"
value="Derouler" onClick="deroule()"></td>
            <td width="6%"><input type="button" name="Submit6"
value="Fuir" onClick="fuit()"></td>
            <td width="10%"><input type="button" name="Submit7"
value="Reduire" onClick="reduit()"></td>
            <td width="11%"><input type="button" name="Submit8"
value="Grossir" onClick="grossis()"></td>
            <td width="8%"><input type="button" name="Submit9"
value="Battre" onClick="battre()"></td>
            <td width="9%"><input type="button" name="Submit10"
```

```
value="Nuage" onClick="nuage()"></td>
            <td width="6%"><input type="button" name="Submit11"
value="Rebouger" onClick="rebouger()"></td>
            <td width="18%"><input type="button" name="Submit12"
value="Retour" onClick="retour()"></td>
          </tr>
        </table>
        </p>
  </div>
  </body>
</html>
```

Ce script s'exécute au chargement de la page. Après avoir chargé les deux fichiers nécessaires aux fonctionnalités de la bibliothèque scriptaculo.us, le script définit une variable permettant d'économiser la saisie pour les manipulations suivantes. L'ensemble des effets appliqués à l'objet peut être lancé par un clic sur un bouton, où figure le nom de l'effet.

La bibliothèque script.aculo.us, en plus de ces effets visuels assez impressionnants, permet de maîtriser facilement des techniques JavaScript évoluées. La première d'entre elles a déjà été détaillée précédemment, mais il convient de comparer la complexité de ce script avec la simplicité d'utilisation offerte par script.aculo.us.

4. Le drag-and-drop avec script.aculo.us

La création de drag-and-drop avec script.aculo.us est d'une facilité déconcertante, d'autant plus que les options permettent de nombreuses d'alternatives. La méthode à utiliser doit suivre deux étapes. Dans un premier temps, il s'agit d'indiquer quels sont les éléments qui peuvent être déplacés, puis dans un second temps, quels sont ceux qui sont désignés pour les recevoir.

La syntaxe à suivre, lors de la désignation des objets à déplacer, est la suivante :

```
new Draggable("id de l'objet à déplacer", {options}) ;
```

Les options disponibles sont les suivantes :

Nom de l'option	Fonction
`constraint: "horizontal" "vertical"`	Limite le déplacement de l'objet dans le sens horizontal ou vertical.
`endeffect:function()`	Indique la fonction éventuelle à exécuter en fin d'effet.
`ghosting: "true" "false"`	Crée et déplace un clone de l'élément.
`handle: "true" "false"`	Indique si une poignée de déplacement doit être utilisée.
`revert: "true" "false"`	Indique si l'élément doit revenir à sa position initiale après avoir relâché la souris.
`reverteffect:`	Indique la fonction éventuelle à exécuter lors du retour de l'élément avec revert.
`snap:[x,y]`	Indique une grille en pixels selon laquelle l'élément se déplace.
`starteffect:`	Indique la fonction éventuelle à exécuter au début de l'effet.
`zindex:(1)`	Indique l'indice de positionnement de l'élément selon l'axe z.

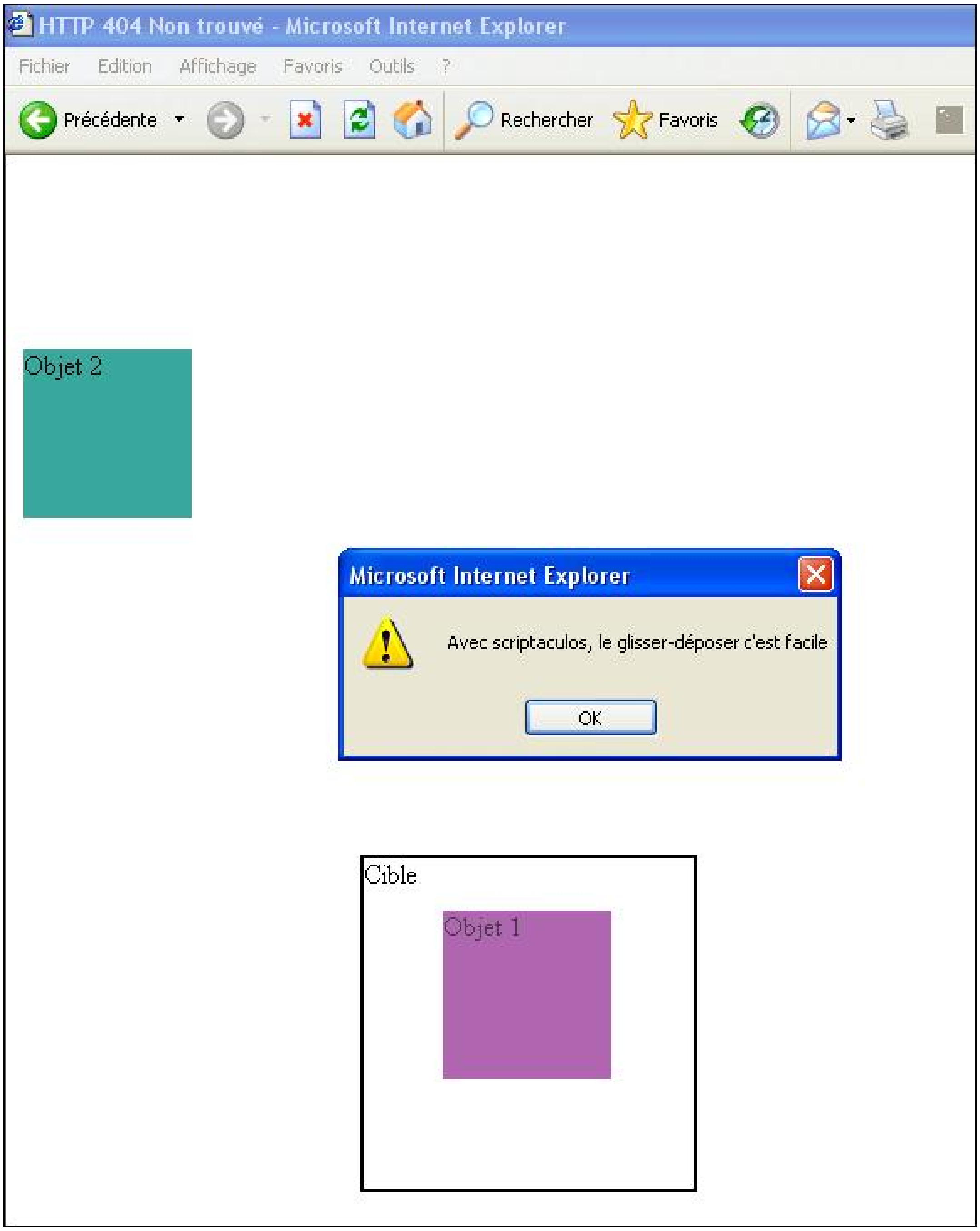
Une fois les éléments à déplacer définis, il faut encore indiquer qui les recevront. Si l'élément attendu est bien déposé dans l'objet cible, il est alors possible d'exécuter un effet visuel ou d'afficher une boîte de dialogue. La syntaxe, permettant de définir un objet cible, est la suivante :

```
droppables.add("id de l'objet à déplacer", {options}) ;
```

Les options disponibles sont les suivantes :

Nom de l'option	**Fonction**
`accept: "nom de la (les) classe (s)" [classe1,classe2]`	Indique les éléments autorisés par la cible. C'est la classe qui correspond à l'objet ou aux objets accepté (s).
`containment: "element" [element1,élément2]`	Indique la cible et les éléments acceptés.
`hoverclass:`	Permet de modifier la classe de la cible lorsqu'un élément passe par-dessus.
`overlap: "horizontal " "vertical"`	Une fonction peut être exécutée si la cible est couverte à plus de 50% de la surface dans la direction spécifiée.
`onHover:fonction(element,cible, pourcentage)`	Exécute la fonction, lorsque l'élément spécifié recouvre la cible avec un pourcentage de recouvrement, passé en paramètre.
`OnDrop:fonction(element,cible)`	Exécute la fonction lorsque l'élément est lâché sur la cible.

Exemple : créer un effet drag-and-drop permettant de déplacer deux calques dans une cible. Si l'objet correspond à la bonne réponse, une boîte de dialogue s'affiche et un effet est appliqué à la cible.




```
<html>
<head>
<script language="javascript" src="prototype.js" >
</script>
<script language="javascript" src="scriptaculous.js">
</script>
</head>
<body>
<style type=text/css>
.objet1 {background-color:#993399;height:100px;width:100px;
z-index:1}
.objet2{background-color:#336666;height:100px;width:100px;
z-index:1}
.cible {border-right:#000000 2px solid; border-top: #000000 2px
solid; border-left:#000000 2px solid;height:200px;width:200px;
left:200px;top:200px;border-bottom:#000000 2px solid}
</style>
<div class=objet1 id=objet1>Objet 1</div>
<div class=objet2 id=objet2>Objet 2</div>
<div class=cible id=cible>Cible</div>
```

```
<script language="javascript">
new Draggable("objet1");
new Draggable("objet2",{revert:true});
Droppables.add("cible", {
accept: "objet1",
onDrop: function() { new Effect.Highlight("cible"); alert("Avec
scriptaculos, le glisser-déposé c'est facile");
new Effect.MoveBy("cible",100,100)}});
</script>
</body>
</html>
```

Ce script s'exécute au chargement de la page. Tout comme précédemment, les deux fichiers de la bibliothèque script.aculo.us sont appelés dès le début. Ensuite, les styles CSS de chaque objet (objet1, objet2 et cible) sont définis et le script permet de déplacer les objets grâce à l'instruction new Draggable. Ici, l'option revert :true indique que l'objet concerné devra reprendre sa position initiale après le déplacement. Le script indique, ensuite, l'objet devant recevoir les autres objets par l'utilisation de la méthode Droppables.add, qui définit les évènements lorsque l'objet est accepté dans la cible. L'évènement onDrop permet de modifier l'apparence de la cible, d'afficher un message dans une boîte de dialogue et de modifier la position de la cible, afin de recommencer.

L'avantage de l'utilisation de script.aculo.us est incontestable, notamment dans le traitement des effets visuels et graphiques, mais cela n'est pas le seul domaine dans lequel cette bibliothèque peut être d'un grand secours. En effet, la mise en place de la technologie AJAX est particulièrement simplifiée.

5. L'autocomplétion avec script.aculo.us

Script.aculos.us met à disposition un objet puissant, permettant de gérer facilement l'autocomplétion (multiples propositions lors d'une saisie), par l'intermédiaire de la technologie AJAX. Il s'agit de l'objet new Ajax.autocompleter, dont la syntaxe est la suivante :

```
new Ajax.Autocompleter("champsuggestions","affichage","url");
```

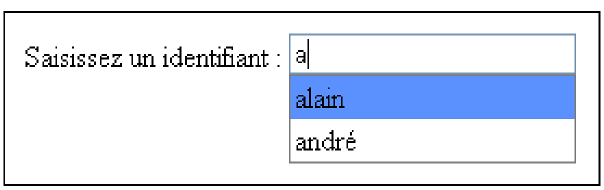
Où champsuggestions correspond au champ d'un formulaire recevant les suggestions, affichage à l'élément devant les afficher et url au fichier (généralement en php) permettant d'extraire et de composer la liste des suggestions.

Les suggestions peuvent être renvoyées par :

- un simple fichier écrit en php comprenant un tableau,
- une base de données (MySQL avec PhP) interrogée par une requête.

Le premier type d'utilisation, même s'il est moins souple, est plus facile.

Exemple 1 : créer un script qui propose une liste de prénoms, correspondant à la saisie effectuée dans un champ de formulaire. Le fichier PhP comprend les valeurs à retourner, sous la forme d'un tableau.



Le script du fichier requetesimple.php :

```
<?php
header('Content-type: text/html; charset=iso-8859-1');
$identifiants = array('Alain','André','Éric','Frédéric',
'Isabelle','Nathalie','Nestor',);
echo "<ul>\n";
foreach ($identifiants as $identifiant){
if (stripos($identifiant, $_POST['identifiant']) === 0){
echo "    <li>$identifiant</li>\n";
}
}
echo "</ul>";
?>
```

La première ligne du script permet d'indiquer que vous travaillez à partir du jeu de caractère 8859-1, permettant de renvoyer les caractères accentués. Ensuite, le script génère un tableau dans lequel les prénoms sont saisis. Puis, chaque valeur du tableau est comparée à la variable renvoyée par la page. Lorsque des éléments peuvent être renvoyés, ils composent une liste grâce à l'utilisation de la balise <li>.

Le script du fichier html :

```
<html>
<head>
<title>Autocompletion avec scriptaculous</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-
8859-15">
<script language="javascript" src="prototype.js" >
</script>
<script language="javascript" src="scriptaculous.js">
</script>
<style type="text/css">
.propositions {position:absolute;background-color:#CCFFFF;border:1px
solid#330066;margin:0px;padding:0px;}
.propositions ul {list-style-type:none;margin:0px;
padding:0px;overflow:auto}
.propositions ul li.selected {background-color:#6699FF;
color:#FFFFFF;font-weight:bold}
.propositions ul li {list-style-type:none;display:block;margin:0px;
padding:2px;cursor:pointer}
</style>
<form action="" method="get" name="form1" id="form1">Saisissez un
identifiant :<input name="identifiant" type="text" id="identifiant"
size="20" maxlength="20" />
<div id="identifiant_propositions" class="propositions"></div>
</form>
<script language="javascript">
new Ajax.Autocompleter("identifiant","identifiant_propositions",
"requetesimple.php",{paramName:'identifiant',minChars:1});
</script>
</body>
</html>
```

Le script débute par l'appel des bibliothèques prototype et script.aculo.us, nécessaires à l'utilisation de l'objet new Ajax.Autocompleter. Ensuite, il est nécessaire de définir plusieurs styles CSS permettant de présenter convenablement la liste des suggestions. Après la définition du champ de formulaire, le script JavaScript utilise l'objet new Ajax.Autocompleter en lui indiquant :

- pour quel champ l'autocomplétion doit être lancée (ici le champ identifiant),
- comment les suggestions devront être présentées (ici le calque identifiant propositions),
- à partir de quel fichier les données seront renvoyées (ici le fichier requetesimple.php).

Le paramètre renvoyé à ce fichier correspond au nom figurant après paramName avec un nombre minimal de caractères (défini ici par minChars:1).

Exemple 2 : créer un script proposant une liste de prénoms, correspondant à la saisie effectuée dans un champ de formulaire. Le fichier PhP sert de support à la requête SQL permettant l'extraction des valeurs correspondant à la chaîne de caractères, saisie dans le fichier autocompletion.html. Celui-ci reste identique, mis à part l'url du fichier qui prend la valeur requetecomplexe.php à la place de requetesimple.php.

Le script du fichier requetecomplexe.php est le suivant :

```
<?php require_once('../../Connections/connex1.php'); ?>
<?php
header('Content-type: text/html; charset=iso-8859-15');
mysql_select_db($database_connex1, $connex1);
$query_Recordset1 = "SELECT Ref_Utilisateur, Identifiant FROM
Utilisateurs WHERE Identifiant LIKE '".$_POST['identifiant']."%'";
$Recordset1 = mysql_query($query_Recordset1, $connex1) or
die(mysql_error());
$result = mysql_query($sql);
echo "<ul id=\"mylist\">\n";
while($data = mysql_fetch_assoc($Recordset1)) {
```

```
echo "<li id=\"item_" . $data['Ref_Utilisateur'] . "\">" .
$data['Identifiant'] .
"</li>\n";
}
echo "</ul>";
?>
<?php
mysql_free_result($Recordset1);
?>
```

Les premières lignes du fichier permettent la connexion à la base de données. Il est important ensuite de définir dans quel jeu de caractères le script doit fonctionner (ici, avec le jeu dont le code est 8859-15 pour interpréter les caractères accentués). Le script permet, ensuite, de calculer le nombre de résultats à renvoyer et de construire la liste des valeurs, grâce à la balise <li>. Ici, la table MySql comprend les mêmes valeurs que le tableau en PhP précédemment créé, le résultat doit donc être identique. L'avantage d'une telle solution repose sur le fait qu'il n'est plus obligatoire de changer, ajouter ou supprimer les valeurs proposées dans le code directement, ce qui simplifie la maintenance de cette fonctionnalité.

Outre les bibliothèques Prototype et Script.aculo.us, il existe beaucoup d'autres bibliothèques avec des fonctionnalités tout aussi intéressantes. Il est possible ainsi de citer la bibliothèque Dojo (disponible à l'adresse http://dojotoolkit.org/downloads) contenant de nombreux outils graphiques comme la représentation en arborescence. Bref, un vaste champ de découvertes et de développement en perspective !